Universität Bremen

Bachelorarbeit

Fachbereich 03
Universität Bremen

Entwicklung einer Client-Server-Lösung für den SEE-Multiplayer-Modus

Thorsten Friedewold, 4510336, thofri@uni-bremen.de
Simon Leykum, 4509142, sleykum@uni-bremen.de

Erstgutachter: Prof. Dr. Rainer Koschke
Zweitgutachterin: Dr. Hui Shi

25. Januar 2024

Abstract

Unsere Arbeit konzentriert sich auf die Entwicklung einer dedizierten Serververwaltungsplattform für SEE Game-Server. Das Hauptziel besteht darin, das Management der einzelnen Game-Server aus SEE in eine benutzerfreundliche Oberfläche zu verlagern, die über einen Webbrowser zugänglich ist. Unsere Arbeit umfasst die Entwicklung von dem erforderlichen Frontend als auch des Backend (mit Anbindung an eine Datenbank und einen Dateispeicher). Zusätzlich erläutern wir die Modifikationen, die wir vorgenommen haben, um SEE in einen eigenständigen Server zu transformieren, um letztendlich alle Dienste mithilfe von Docker zu betreiben. Abschließend Evaluieren wir die Entwicklungen um einschätzen zu können welche Maßnahmen der Entwicklung Sinnvoll waren. Unsere Ergebnisse zeigen eine durchaus positive Entwicklung, wir waren in der Lage die von uns geplanten Funktionen zu Implementieren.

Hinweis: Autoren

Da diese Bachelorarbeit von zwei Autoren verfasst wurde, wird der jeweilige Autor jeweils am Anfang eines Abschnittes in fetter Schrift vermerkt. Falls am Anfang eines Abschnittes kein Autor vermerkt ist, ist davon auszugehen, dass der Autor des vorherigen Abschnitts auch diesen verfasst hat.

Erklärungen

Offizielle Erklärungen von

Thorsten Friedewold, Matrikelnr. 4510336
Simon Leykum, Matrikelnr. 4509142

Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht. Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:
1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach u. Jahr.

Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Mit meiner Unterschrift versichere ich, dass ich die oben stehenden Erklärungen gelesen und verstanden habe. Mit meiner Unterschrift bestätige ich die Richtigkeit der oben gemachten Angaben.

Bremen, den 25.01.2024

(Thorsten Friedewold)	(Simon Leykum)

Inhaltsverzeichnis

1 Glossar

Glossar

Backend Eine Serversoftware, die Anfragen von Clients entgegennimmt. 2, 4, 6, 9–11, 19–23, 25, 27, 31, 32, 34, 35, 37–39, 47, 48, 69–75

Client Eine Einzelne Instanz, welche mit einem Server kommuniziert. 6–8, 11, 24, 41–48, 50, 72–75

Code Textuelle Instruktionen für einen Computer um etwas auszuführen. 6, 25, 42

Code-City Die visuelle darstellung von Code als 3-Dimensionale Blöcke, ähnlich einer Stadt. 4, 7, 24, 25, 45, 47, 75

Component Ein ReactJS konzept. Ein Component ist ein wiederverwendbarer Codeabschnitt. 25–28, 30, 31

Cookie Eine Textinformation welche von einem Webbrowser für einen Webseite gespeichert wird. 27

Dateispeicher Ein Server, der Daten als Datei speichert und diese einfach wieder abrufbar machen kann (in unserem Fall MinIO). 2, 10, 11, 71, 72

Datenbank Ein Server, der Daten speichert und diese Mittels bestimmter Befehle löschen, filtern, bearbeiten, oder hinzufügen kann. 2, 7, 10, 11, 20, 32, 38, 39, 71, 72

Docker Eine Plattform um Anwendungen innerhalb eines simulierten Computers auszuführen. 2, 11, 20, 72–74

Frame Ein Bild. 39

Framework Ein Rahmenwerk zur Entwicklung von Software. 6, 7, 18, 19

Frontend Teil einer Web-App, welche die Visuelle Darstellung der Benutzeroberfläche und die Kommunikation zwischen Benutereingabe und Backend abwickelt. 2, 4, 7, 9–11, 18, 19, 25, 27, 29, 31, 55, 65, 67, 69, 72–74

Game-Engine Ein Framework zur entwicklung von Videospielen. 7

Game-Server Eine Art eines Servers welche im Kontext von Videospielen verwendet wird. 2, 7, 9–14, 19, 20, 25, 29, 34, 35, 37–39, 47, 50, 51, 53, 56, 69, 70, 72–76

GameObject Ein Unity-Konzept welches ein Objekt innerhalb der Spielwelt definiert. 45, 75

Host Eine Instanz welche sowohl Client als auch Server ist. 7, 8, 24, 45

HTML HyperText Markup Language. Eine Sprache zum Strukturieren von Webseiten. 31

Javascript Eine Skriptsprache. 7, 19

Klasse Ein Code-Modell eines Objektes. 39, 45

Library Eine gebündelte Sammlung an wiederverwendbarem Code. 19, 24, 26, 27, 39, 42, 45

Open-Source Der Quellcode einer Open-Source Software ist für die Allgemeinheit einsehbar. 19

React Ein Frontend-Framework. 25, 28

REST-API Eine Schnittstelle zwischen Datenbank und Frontend welche Anfragen entgegennimmt und hierauf Antworten sendet. 19, 20

RPC Remote Procedure Call. 42, 45

SEE SEE (Software Engineering Experience) ist eine Software entwickelt von der Uni Bremen in Kooperation mit der Axivion GmbH für kollaborative Softwareanalyse aufgrund der Code-City metapher. 2, 7, 8, 23–25, 39, 49

Server Eine Instanz, welche mit vielen Clients kommuniziert. 2, 6, 7, 11, 13–15, 23–25, 38–43, 45–47, 49, 50, 73, 75

Software Programme die auf einem Computer ausgeführt werden können. 7, 24–26, 68, 69

SUS System Usability Scale. Eine Methode um die Benutzbarkeit einer Software zu analysieren. 55, 66–68

Typescript Eine Skriptsprache mit Typisierung. Eine Erweiterung von Javascript. 19, 25

UI User Interface. Die Visuelle Darstellung der Benutzeroberfläche. 11

Unity Eine Game-Engine, entwickelt von Unity Technologies. 6, 24, 39, 40, 43, 46

URL Uniform Resource Locator. Ein Link zu einer Webresource. 26, 27, 48

UX User Experience. Wie eine Benutzer*in mit der Software interagiert und deren Erfahrung hierbei. 11

Web-App Eine Anwendung welche durch einen Webbrowser aufgerufen werden kann. 6, 18

2 Einleitung

2.1 Motivation

Thorsten Friedewold:
SEE beschreibt sich selbst als "Collaborative Software"[Kos23], mit der man Analysen von Quellcode visualisieren kann. Im Verlauf unseres Bachelorprojekts, in dem wir SEE kennengelernt haben, kam uns häufig der Gedanke, dass dieser kollaborative Ansatz weiter ausgebaut werden könnte. Unsere Idee beschäftigte sich hauptsächlich mit einer separaten Verwaltungssoftware für dedizierte Meetingräume. Im Zuge der Bachelorarbeit kamen wir auf diese Idee zurück. Sie erschien uns passend, da wir bereits in anderen eigenen Projekten angewandte Lösungen hier ansetzen konnten. Unser Ziel war es, für SEE eine Plattform zu schaffen, über die Game-Server einfach verwaltet werden können.

2.2 Aktueller Stand in SEE

Simon Leykum:
Zum aktuellen Zeitpunkt funktioniert der Multiplayer-Modus in SEE durch eine Client-Host-Architektur. Wobei sich mehrere Clients mit einem Host verbinden. Der Host ist Server und Client in einem. Das heißt also, dass immer eine Spieler*in Host sein muss, die Netzwerkkommunikation läuft dann über die Instanz dieser Spieler*in.

Aufgrund der aktuellen Client-Host-Architektur ist es also nicht möglich eine dedizierte Server-Instanz zu erstellen. Dieses und der Fakt, dass immer eine Spieler*in Host sein muss führt zu mehreren Problemen.

Zum Ersten ist dies aus sicherheitstechnischer Sicht nicht ideal, da die Instanz, die die Daten empfängt und an die anderen Spieler*innen weitergibt, ebenso eine Spieler*in ist. Hierdurch könnte diese Spieler*in die Daten beliebig manipulieren. Auch muss die Person, die Host ist, ihr eigenes Netzwerk freigeben um die Informationen der anderen Spieler*innen zu empfangen. Dieses bringt weitere Sicherheitsrisiken mit sich.

Auch gibt es keine Möglichkeit den Host zu ändern, sodass die Spieler*in welche Host ist, sich im Laufe der Sitzung nicht von dieser trennen kann, ohne dass diese abbricht.

In einem Unternehmenskontext wäre es zudem gut, wenn es möglich wäre, eine Instanz dauerhaft am Laufen zu haben, sodass Spieler*innen beliebig dieser beitreten können, sodass sich Mitarbeiter*innen in SEE treffen können, ohne, dass dies zunächst mühsam vorher abgesprochen werden muss. Dieses ist zwar aktuell insofern möglich, dass eine Host-Instanz gestartet und angelassen werden könnte, dies führt aber zu einigen Problemen. Zum einen wird hierbei immer ein Avatar in die Welt gesetzt, zum anderen gibt es keine Möglichkeit diese Instanz einfach zu verwalten. Auch ist es aktuell nicht möglich den aktuellen Spielstand an Clients zu schicken welche sich später mit dem Host verbinden.

2.3 Angestrebte Funktionalität

Thorsten Friedewold, Simon Leykum:
Aufgrund der im vorherigen Abschnitt genannten Punkte haben wir uns entschieden die aktuelle Client-Host-Architektur von SEE zu erneuern und um eine dedizierte Serverlösung zu erweitern. Hierzu soll zudem eine Weboberfläche entwickelt werden, über welche es möglich sein soll verschiedene Instanzen einfach zu verwalten. Auf die engeren Details der Planung und Umsetzung gehen wir weiter in den kommenden Abschnitten ein. Hier wollen wir erstmal eine Übersicht über die angestrebten Funktionalitäten unserer Lösung geben.

Priorität	Erstrebte Implementierung	Aktuell Implementiert	Beschreibung
1	Allgemeine Multiplayer Funktionalität	Ja	Die Möglichkeit mit anderen Personen gemeinsam eine Code-City zu betrachten
1	Aktuellen Stand der Code-City an Benutzer*innen senden	Nein	Wenn eine Benutzer*in nachträglich einem Meetingraum beitritt, sollte diese den aktuellen Stand der Code-City übermittelt bekommen
1	Dedizierte Serverlösung	Nein	Eine ausgelagerte Serverlösung, um das Hosten von Räumen zu vereinfachen und zentral zu verwalten
2	Web-UI für Server	Nein	Eine Web-UI, um den dedizierten Server zu verwalten; hier könnte zum Beispiel angezeigt werden, welche Code-City geladen ist, oder wie viele Personen sich gerade in einem Meetingraum befinden
3	Managen von mehreren Meeting-Räumen	Nein	Die Möglichkeit, unter einer Server-Umgebung mittels einer Web-UI mehrere verschiedene Meeting-Räume anzulegen und zu verwalten
3	Snapshots für Änderungen	Nein	Speichern von aktuellen Ständen einer Code-City innerhalb eines Meetingraums
4	Automatisches Herunterladen von Code-Cities auf Client-Rechner	Nein	Nutzer*in sollen die in einem Meetingraum verwendeten Code-Cities automatisch herunterladen können
5	Benutzerautorisierung	Nein	Nutzer*innen soll es möglich sein, Meetingräume nur mittels eines Schlüssels zu betreten, um unautorisierten Zugriff zu verhindern

Tabelle 1: Angestrebte Funktionen im Vergleich zum aktuellen Entwicklungsstand

3 Konzeption

In diesem Abschnitt möchten wir in genauerem Detail auf die Planung von Frontend, Backend, dem Game-Server, und der allgemeinen Architektur unserer Lösung eingehen. Insgesamt ist es uns wichtig die Entwicklungsziele für das Frontend, Backend und den Game-Server zu definieren und einen Plan zu erstellen um diese umzusetzen.

3.1 Architektur

Thorsten Friedewold:
Der Einstieg in unsere Konzeption soll ein grober Entwurf der von uns angestrebten Architektur sein. Dabei möchten wir auf die einzelnen Komponenten eingehen, die wir für die Funktion unserer Lösung benötigen.

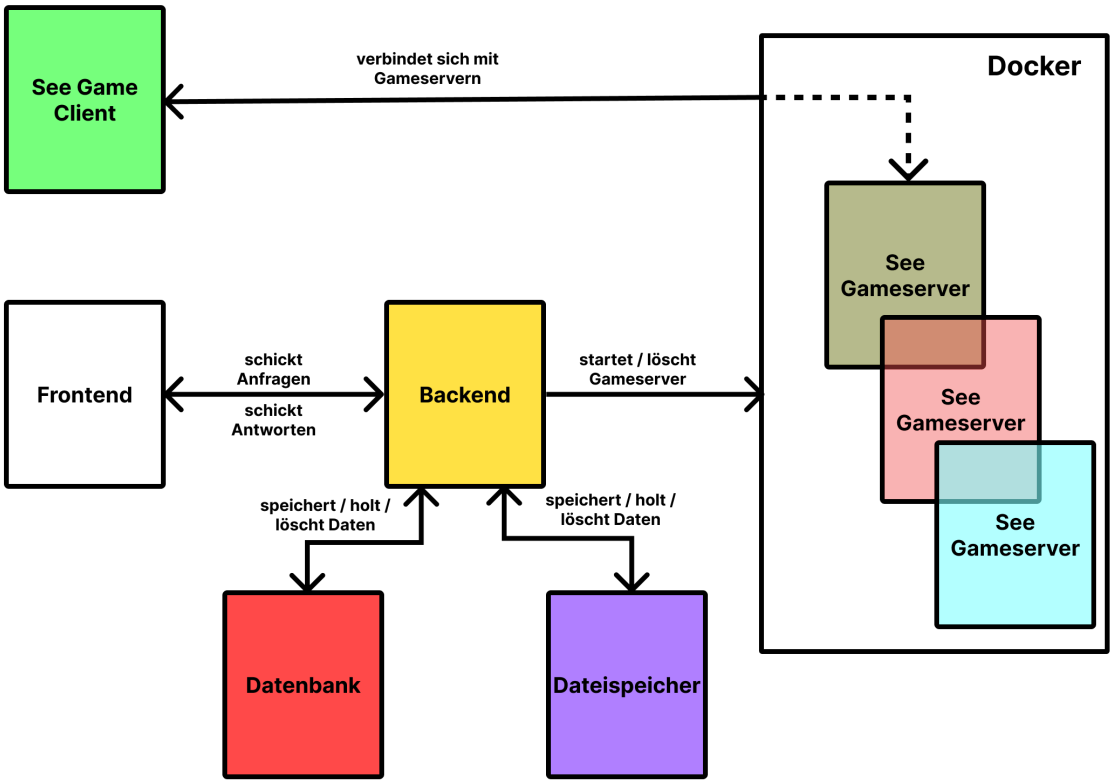

Abbildung 1: Angestrebte Architektur

Wie zu erkennen ist, besteht unsere angestrebte Architektur aus den folgenden Komponenten:

- Frontend
- Backend
- Game-Server
- Datenbank
- Dateispeicher

Das Frontend soll über das Backend Game-Server starten können und diese anschließend für die Nutzer*innen darstellen. Um mehrere Game-Server starten zu können, möchten wir diese in ein Docker-Image überführen, um diese über das Backend starten und beenden zu können. Danach soll es Clients möglich

sein, sich mit einem der gestarteten Game-Server zu verbinden. Die Datenbank soll Informationen über die Server und Nutzer*innen speichern, während der Dateispeicher die für den Game-Server relevanten Dateien speichern soll.

3.2 Frontend

Simon Leykum:

Die Planung des Frontends teilt sich im Wesentlichen in zwei Abschnitte auf. Zum Einen die Planung der Gestaltung der Benutzeroberfläche (UI) und des Nutzererlebnisses (UX) und zum Anderen die Planung der technischen Umsetzung dieser.

3.2.1 UI und UX

Um die Gestaltung der Benutzeroberfläche und des Nutzerlebnisses zu planen haben wir zunächst einen Prototypen in *Figma* erstellt. *Figma* ist eine kollaborative Anwendung um unter anderem Prototypen für Webseiten, Apps und weitere digitale Produkte zu erstellen.[Wha]

Bei dem Aufrufen des Frontends wird der Benutzer*in zunächst ein Anmeldeformular angezeigt. Hier kann die Benutzer*in ihren Benutzernamen und ihr Passwort angeben und sich anmelden.

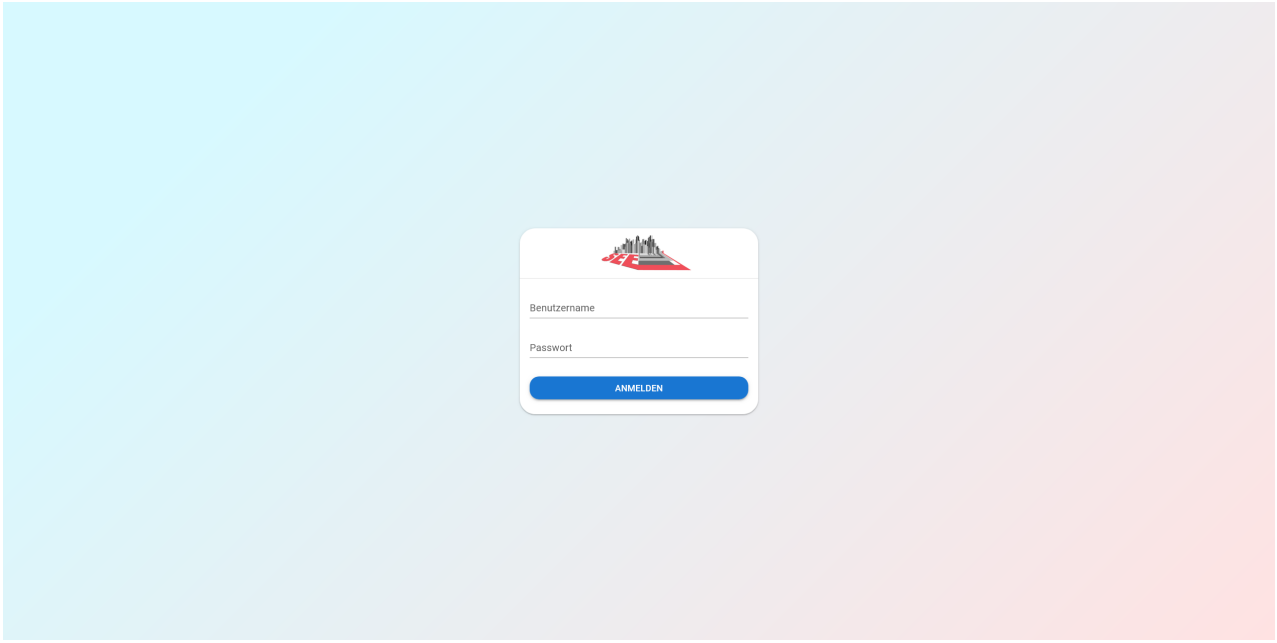

Abbildung 2: Anmelde-Ansicht

Nach dem Anmelden soll die Benutzer*in auf eine Seite weitergeleitet werden, auf welcher sie eine Übersicht über alle erstellten Game-Server und deren Status bekommen soll.

Auf allen Seiten, nachdem die Benutzer*in sich angemeldet hat, befindet sich oben auf jeder Seite eine Kopfzeile. Auf der linken Seite dieser Kopfzeile ist das SEE-Logo zu sehen, wenn die Benutzer*in auf dieses Logo klickt, wird sie zu der Übersichtsseite weitergeleitet. Auf der rechten Seite sind drei Knöpfe. Der linke Knopf leitet die Benutzer*in zu der allgemeinen Einstellungsseite weiter. Dieser Knopf wird nur angezeigt,

wenn die Benutzer*in Administrator ist. Durch den mittleren Knopf wir die Benutzer*in auf die persönliche Einstellungsseite weitergeleitet. Mithilfe des rechten Knopfs kann die Benutzer*in sich abmelden.

Auf der Übersichtsseite werden unterhalb der Kopfzeile die erstellten Gameserver als Kacheln in einer Liste angezeigt. Auf der linken Seite einer Kachel ist das Server-Bild zu sehen, Ziel des Server-Bilds ist es, die Server auf einen Blick auseinanderzuhalten und wiedererkennen zu können. In der Mitte der Kachel ist unter dem Namen des Game-Servers der aktuelle Status durch einen Banner einzusehen, hierdurch ist es möglich, direkt zu erkennen, welche Game-Server offline und welche Game-Server online sind. Um diese Information zu unterstützen, wird zudem angezeigt, seit wann der Game-Server bereits online bzw. offline ist. Zuletzt ist auf der rechten Seite der Kachel ein Knopf, welcher es erlaubt, die *IP-Adresse* des Servers zu teilen. Die IP-Adresse ist notwendig um sich später mit dem Game-Server verbinden zu können.

Unterhalb der Liste der Game-Server befindet sich ein Knopf, über welchen es möglich ist, weitere Gameserver hinzuzufügen.

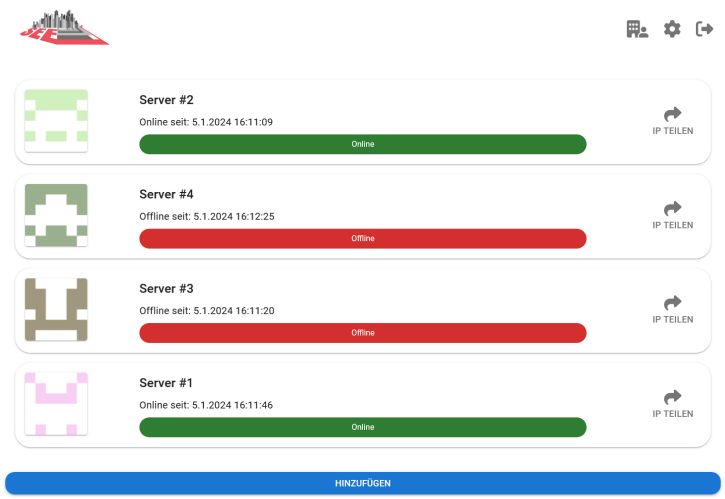

Abbildung 3: Übersicht-Ansicht

Nach dem betätigen dieses Knopfes wird die Benutzer*in auf eine Seite weitergeleitet, auf welcher sie einen neuen Game-Server erstellen kann.

Um einen Game-Server zu erstellen, kann die Benutzer*in hier zunächst den Namen des Servers angeben. Auch ist es möglich, ein Serverpasswort zu setzen, um den Game-Server von unbefugten Personen zu schützen. Neben diesen Einstellungen wird das automatisch generierte Server-Bild angezeigt, falls dieses der Benutzer*in nicht gefällt, ist es möglich, mithilfe eines Klicks auf das Server-Bild dieses neu zu generieren.

Unterhalb der Game-Server-Einstellungen ist es möglich, die Projektdateien, welche für die Game-Server-Instanz benötigt werden, hochzuladen.

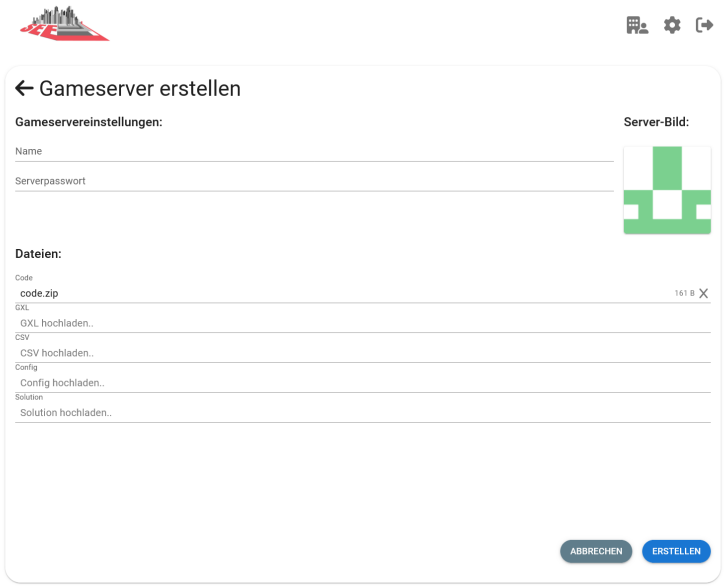

Abbildung 4: Server-Erstellen-Ansicht

Nach dem Erstellen des Game-Servers, oder dem Abbrechen des Erstellens, wird die Benutzer*in wieder zurück auf die Übersichtsseite weitergeleitet.

Beim Klicken auf eine der Server-Kacheln auf der Übersichtsseite wird die Benutzer*in auf die Detailansicht des jeweiligen Servers weitergeleitet.

Diese Ansicht beinhaltet zunächst die gleichen Informationen und Funktionen, die auch in der Übersicht zu sehen waren.
Zusätzlich hierzu ist es aber noch möglich, den Server zu starten bzw. zu stoppen und den Server zu löschen. Des Weiteren ist es möglich, das eingestellte Serverpasswort einzusehen. Dieses ist standardmäßig verdeckt, kann aber durch einen Klick auf den nebengelegenen Knopf sichtbar gemacht werden. Auch kann hier eingesehen werden, welche Dateien vorher hochgeladen worden sind und es ist möglich, über einen Knopf diese Dateien zu Backupzwecken wieder herunterzuladen.

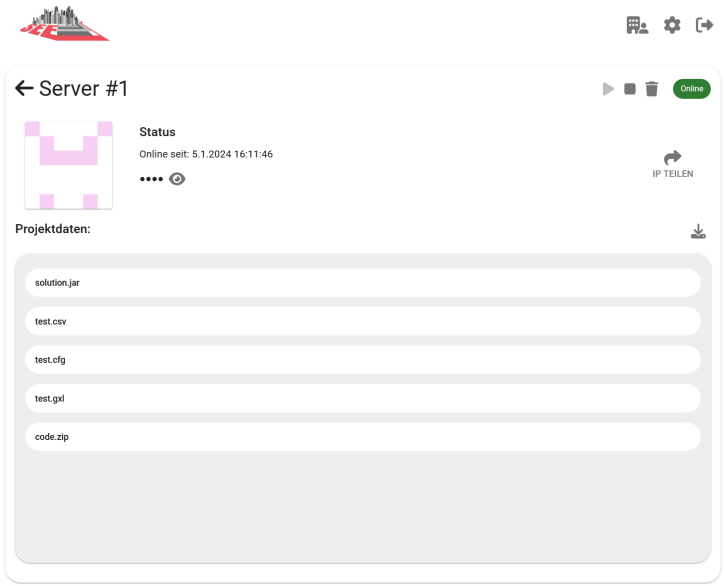

Abbildung 5: Allgemeine-Einstellungen-Ansicht

Wie bereits zuvor beschrieben, befindet sich nach der Anmeldung der Benutzer*in auf jeder Seite eine Kopfzeile, von welcher es möglich ist, auf die allgemeine Einstellungsseite und auf die persönliche Einstellungsseite zu gelangen.

Auf der allgemeinen Einstellungsseite befindet sich die Benutzer*inneverwaltung.

Hier soll eine Liste der hinzugefügten Benutzer*innen angezeigt werden. Über diese Liste soll es möglich sein, diese Benutzer*innen wieder von der Organisation zu entfernen. Über ein Klicken auf das Kronensymbol auf der rechten Seite der Listenelemente soll es zudem möglich sein, eine Benutzer*in als Admin zu ernennen oder ihr den Admin-Status wieder abzuerkennen.

Unterhalb der Liste befindet sich noch ein Knopf, über welchen es möglich ist, neue Benutzer*innen zu der Organisation hinzuzufügen.

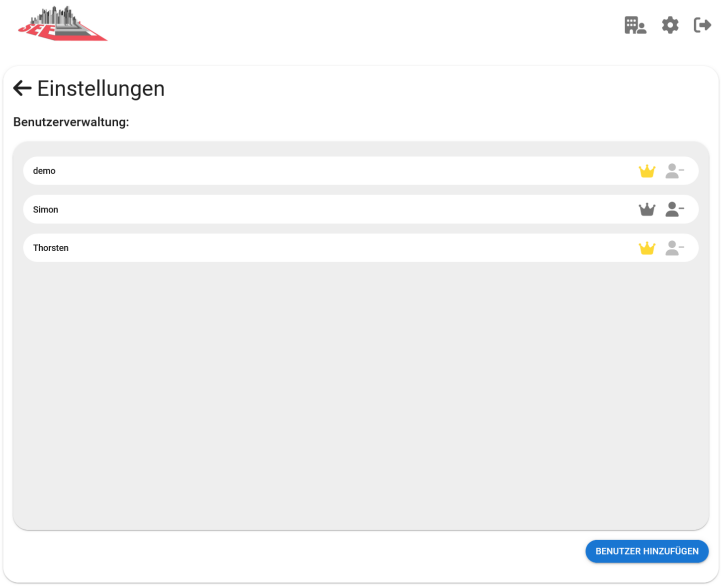

Abbildung 6: Allgemeine-Einstellungen-Ansicht

Bei dem Klicken auf den Knopf mit dem Piktogramm, welches ein Zahnrad darstellt, soll die Benutzer*in auf die persönliche Einstellungsseite weitergeleitet werden.

Auf dieser Seite soll es möglich sein, den eigenen Benutzernamen und das eigene Passwort anzupassen.

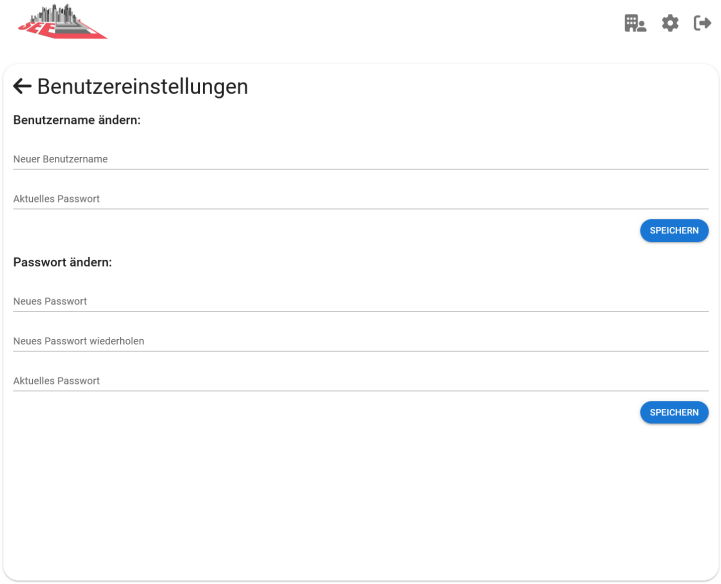

Abbildung 7: Allgemeine-Einstellungen-Ansicht

Im Allgemeinen waren unsere Ziele bei der Gestaltungen des Frontends für eine einfache, intuitive Bedienbarkeit zu sorgen und ein visuell-ansprechendes, professionelles Design zu entwickeln. Durch die

einfache Bedienbarkeit und das einfache Design soll zudem eine gewisse Barrierefreiheit ermöglicht werden. Somit haben wir auch darauf geachtet, dass genügend Farbkontrast zwischen Texten und Hintergründen besteht und dass Texte und Knöpfe ausreichend groß sind, sodass diese einfach zu erkennen und zu bedienen sind.

3.2.2 Technische Umsetzung

Um die technische Umsetzung des Frontends zu planen, haben wir uns zunächst mit verschiedenen Technologien für die Frontendentwicklung von Web-Apps beschäftigt. Hierbei gab es einige Frameworks zur Auswahl. Aufgrund von Vorerfahrungen haben wir uns dazu entschieden, *ReactJS* zu verwenden.

ReactJS, auch *React* ist ein Open-Source-Frontend-Framework, entwickelt von *Meta* und der Open-Source-Community.[Rea]

Zudem haben wir uns entschieden, Typescript zu verwenden. Typescript ist eine Erweiterung der Skriptsprache Javascript. Eine Skriptsprache wird in der Webentwicklung benötigt, um im Browser Code ausführen zu können, damit die Seite nicht nur statisch da steht.
Der Vorteil von Typescript gegenüber Javascript ist die Typisierung, hierdurch kann Variablen ein Typ zugewiesen werden (z.B. `string`; eine Zeichenkette oder `number`; eine Zahl). Hierdurch kann es einfacher sein, sich in den Code eines Tools hineinzuarbeiten und durch das Verwenden von Typescript können möglicherweise Fehler verhindert bzw. verringert werden.

Zusätzlich zu den vorher genannten Technologien verwenden wir einige weitere Technologien, konkret *pnpm* zum Verwalten von externen Libraries und *vite* zum lokalen Starten des Frontends.

Des Weiteren verwenden wir einige externe Libraries für *React*, eine wesentliche hiervon ist *Material UI*, dies ist eine Library welche viele vorgefertigte Bausteine für das Bauen der Benutzeroberfläche bietet.

3.3 Backend

Thorsten Friedewold:

Das Backend hat zum Ziel, die Schnittstelle zwischen dem Frontend und dem Game-Server darzustellen und dabei zusätzliche Daten zu speichern, um den Nutzer*innen Informationen über die verschiedenen Game-Server bereitzustellen.

3.3.1 Entwicklungsziele

Im Folgenden sind die Entwicklungsziele für das Backend festgehalten.

- Kommunikation zwischen Backend und Frontend über eine REST-API.
- Nutzer*innen sollen über das Frontend Game-Server erstellen und verwalten können.
- Speicherung von Projektdaten für den Game-Server im Backend.
- Erstellung und Verwaltung von einem oder mehreren Game-Servern durch das Backend.
- Speicherung von Informationen über Nutzer*innen und Game-Server im Backend.
- Bereitstellung von Autorisierung und Authentifizierung für Nutzer*innen durch das Backend.

3.3.2 Datenbankmodell

Für das Backend haben wir folgende wesentlichen Entitäten herausgearbeitet, die wir für die Implementierung benötigen.

- Server: Stellt einen Game-Server dar
- User: Stellt eine Nutzer*in dar.
- File: Stellt eine Datei dar, die einem Server zugewiesen wird.
- Rolle: Stellt eine Rolle für einen User dar, damit werden die Rechte verwaltet.

Wir wollen im Abschnitt Implementierung genauer auf die einzelnen Entitäten eingehen (siehe Abschnitt 4.2).

3.3.3 Mögliche Anfragen an das Backend

Als Grundlage für die Entwicklung haben wir uns entschieden, Spring Boot [1] zu verwenden. Spring Boot ist ein Open-Source-Java-Framework, das viele Funktionen modular hinzufügen kann, darunter Datenbankanbindungen, REST-APIs und Sicherheitsfeatures. Wir haben bereits Erfahrungen mit diesem Framework gesammelt und es deshalb als Grundlage für unsere Entwicklung gewählt.

Für die Speicherung von Daten für den Game-Server setzen wir auf MinIO [2], ein Open-Source-Projekt, das die Speicherung von Daten mittels einer API ermöglicht. Diese Entscheidung treffen wir, um möglichen Problemen bezüglich Zugriffsrechten auf dem Backend-Server vorzubeugen. Mögliche Probleme in diesem Zusammenhang könnten unter anderem sein, das wir das Backend auf einem Computer betreiben, der es nicht erlaubt Dateien zu speichern.

Die Speicherung von Daten über Nutzer*innen und Game-Server realisieren wir durch eine einfache Datenbankanbindung zwischen dem Backend und einem Datenbank-Server.

Die Erstellung von mehreren Game-Servern und deren Verwaltung lösen wir, indem wir den Game-Server in einen Docker-Container überführen. Dadurch können wir mehrere Game-Server einfach erstellen und verwalten.

Um all diese Funktionen zu ermöglichen, haben wir folgende mögliche Anfragen an das Backend definiert:

Art	**Endpoint**	**Request**	**Erwarteter Response**	**Erklärung**
GET	/server/	UUID serverID	Ein Server mit der ID serverID	Das Backend sucht in der Datenbank nach einem Server mit der ID serverID und gibt diesen zurück, wenn er gefunden werden kann.
GET	/server/all	keine	Alle Server in der Datenbank	Alle Server, die in der Datenbank gespeichert werden, werden zurückgegeben.

[1] https://spring.io/projects/spring-boot/
[2] https://min.io/

POST	/server/create	Server server	Eine gespeicherte Version des übergebenen Servers	Wir geben dem Backend einen neuen Server. Dieser wird dann vom Backend in der Datenbank gespeichert. Beim Speichern erhält der Server eine ID, und das Backend gibt die gespeicherte Version zurück.
POST	/server/addFile	UUID serverID String fileType Multipartfile file	Eine gespeicherte Version der übergebenen Datei	Dem Server mit der ID serverID wird eine Datei file zugeordnet. Das Backend gibt die gespeicherte Datei dann zurück.
DELETE	/server/delete	UUID serverID	Ja, wenn der Server gelöscht wurde, nein sonst	Das Backend löscht den Server mit der ID serverID, sowie alle mit ihm verbundenen Dateien
GET	/server/files	UUID serverID	Liste der dem Server zugehörigen Dateien	Das Backend gibt alle Dateien, die dem Server mit der ID serverID zugeordnet sind, zurück.
GET	/user/me	keine	Der eigene Nutzer*in	Für die Nutzer*in wird anhand des übergebenen JWT die eigene Nutzer*in zurückgegeben
GET	/user/all	keine	Alle Nutzer*innen, die in der Datenbank gespeichert sind	Das Backend gibt alle in der Datenbank gespeicherten Nutzer*innen zurück

POST	/user/create	SignupRequest signupRequest	Eine gespeicherte Version der übergebenen Nutzer*in	Wir geben dem Backend eine neuen Nutzer*in. Diese wird dann vom Backend in der Datenbank gespeichert. Beim Speichern erhält die Nutzer*in eine ID, und das Backend gibt die gespeicherte Version zurück.
POST	/user /addRoleToUser	String username ERole role	Nutzer*in mit überarbeiteten Rollen	Der Nutzer*in wird eine Rolle hinzugefügt.
DELETE	/user /removeRoleFromUser	String username ERole role	Nutzer*in mit überarbeiteten Rollen	Der Nutzer*in wird eine Rolle entfernt.
DELETE	/user/delete	String username	Ja, wenn die Nutzer*in gelöscht wurde, nein sonst	Die Nutzer*in mit dem Username username wird vom Backend aus der Datenbank gelöscht.
PUT	/user /changeUsername	ChangeUsernameRequest changeUsernameRequest	Neuer Cookie für die Nutzer*in	Die Nutzer*in mit dem Username username bekommt einen neuen Username, das Backend generiert daraufhin einen neuen JWT, damit neue Anfragen den neuen Username enthalten
PUT	/user /changePassword	ChangePasswordRequest changePasswordRequest	Ja, wenn das Passwort geändert werden konnte, nein sonst	Die Nutzer*in mit dem Username username erhält ein neues Passwort. Das Backend speichert dieses, aber sendet nichts zurück, außer ja oder nein.

POST	/user/signin	LoginRequest loginRequest	JWT	Die Nutzer*in schickt ihren Username an das Backend. Wenn die Daten stimmen, bekommt die Nutzer*in einen JWT, um sich bei späteren Anfragen zu authentifizieren und zu autorisieren
POST	/user/signout	keine	JWT	Die Nutzer*in bekommt einen leeren JWT zurück, damit Anfragen nicht mehr funktionieren
GET	/file/get	UUID fileID	Datei mit der ID fileID	Das Backend sendet die Datei mit der ID fileID zurück
GET	/file /client/gxl	UUID serverID, String roomPassword	GXL-Datei, die dem Server mit der ID serverID hinterlegt wurde.	Das Backend sendet die Datei, die beim Erstellen des Servers mit der ID serverID als *GXL* hinterlegt wurde zurück.
GET	/file /client/source	UUID serverID, String roomPassword	Source-Datei, die dem Server mit der ID serverID hinterlegt wurde.	Das Backend sendet die Datei, die beim Erstellen des Servers mit der ID serverID als *Source* hinterlegt wurde zurück.
GET	/file /client/config	UUID serverID, String roomPassword	Config-Datei, die dem Server mit der ID serverID hinterlegt wurde.	Das Backend sendet die Datei, die beim Erstellen des Servers mit der ID serverID als *Config* hinterlegt wurde zurück.
GET	/file /client/csv	UUID serverID, String roomPassword	CSV-Datei, die dem Server mit der ID serverID hinterlegt wurde.	Das Backend sendet die Datei, die beim Erstellen des Servers mit der ID serverID als *CSV* hinterlegt wurde zurück.

GET	/file /client/solution	UUID serverID, String roomPassword	Solution-Datei, die dem Server mit der ID serverID hinterlegt wurde.	Das Backend sendet die Datei, die beim Erstellen des Servers mit der ID serverID als *Solution* hinterlegt wurde zurück.

3.4 Game-Server

Thorsten Friedewold, Simon Leykum:

Unser Ziel bei der Planung des Game-Servers war es, die Netzwerkstruktur von SEE insofern anzupassen, dass unsere dedizierte Server-Lösung möglich gemacht wird.
Um dies zu erreichen, haben wir verschiedene möglichkeiten für die Implementierung des Game-Servers abgewogen.

3.4.1 Aktueller Stand

Simon Leykum:

Zum aktuellen Zeitpunkt werden in der Netzwerkstruktur von SEE zwei verschiedene Technologien verwendet, um die Kommunikation zwischen Client und Server zu ermöglichen.

Um die *Player-Actions*, also die Aktionen der Spieler*innen zwischen Client und Server zu versenden, wird die Library *NetworkComms.Net* verwendet. Diese Library bietet eine vielzahl an Möglichkeiten, um Kommunikation zwischen Clients und Servern zu ermöglichen.[Netb]
Neben einer klassischen *Client-Server-Architektur*, in welcher Clients mit einem zentralen Server kommunizieren, bietet *NetworkComms.Net* unter anderem zum Beispiel auch die Möglichkeit einer *Peer-to-Peer*-Architektur in welcher Clients direkt miteinander kommunizieren.[Netb]

Für alles außer der *Player-Actions* wird die Unity-Library *Netcode for GameObjects* verwendet. Hierunter fällt unter anderem das Synchronisieren der Bewegung der Charaktere in SEE und der Voice-Chat.

Netcode for GameObjects wurde zuerst am 27.06.2022 veröffentlicht.[Nfga] und ist eine Library die Kommunikation zwischen Unity-Clients und Unity-Servern ermöglicht.
Ein wesentlicher Vorteil von *Netcode for GameObjects* ist, dass diese Library durch die direkte Einbindung in die Unity-Infrastruktur es ermöglicht, einfacher Unity-Interne Daten zu synchronisieren.

Aufgrund der aktuellen Netzwerkstruktur ist zurzeit auch nur eine Client-Host-Archiktetur möglich, wobei die Instanz der Software auf, welcher der Server läuft auch immer zugleich einen Client beinhaltet.
Aufgrund dieser Einschränkung ist es zurzeit nicht möglich, nur einen Server zu starten.

3.4.2 Entwicklungsziele

Simon Leykum:

Um die Netzwerkstruktur von SEE zu vereinheitlichen, haben wir uns entschieden, die Library *NetworkComms.Net* zu ersetzen und für die gesamte Netzwerkstruktur auf *Netcode for GameObjects* zurückzugreifen.

Das Vereinheitlichen der Netzwerkstruktur und somit das Ersetzen der externen Library bringt vielerlei Vorteile mit sich.
Ein wesentlicher Aspekt ist, dass die Vereinheitlichung die Netzwerkstruktur einfacher machen würde und somit auch die Instandhaltung und Weiterentwicklung der Software erleichtert.
Wie in dem vorhergehenden Teil bereits beschrieben, bringt die direkte Einbindung in die Unity-Infrastruktur Vorteile gegenüber der externen Netwerklösung.
Des weiteren ist durch die Vereinheitlichung das Projekt nicht mehr auf die externe Instandhaltung der *NetworkComms.Net* Library angewiesen. Diese wird schon seit 2016 nicht mehr aktiv instand gehalten.[Neta]
Zuletzt wird dadurch, dass die Unity-Lösung einen wesentlich höheren Abstraktionsgrad hat, die Implementierung der Netzwerkstruktur vereinfacht, sodass viele technische Voraussetzungen, um das Versenden von Informationen möglich zu machen, bereits durch die Unity-Lösung implementiert sind und somit nicht extra entwickelt und instand gehalten werden müssen.

Neben der Vereinheitlichung der Netzwerkstruktur wollen wir erreichen, dass es möglich ist, nur eine Server-Instanz zu starten.
Nach der Vereinheitlichung der Netzwerkstruktur und somit dem Ersetzen der *NetworkComms.Net* Library sollte dies mit wenigen Anpassungen erreichbar sein.

Des Weiteren soll es möglich sein, dass bei Verbinden des Clients alle nötigen Dateien der Code-City, welche in der Server-Instanz geladen ist, zu laden. Infolgedessen soll es möglich sein, den aktuellen Stand der Code-City an den Client zu senden.
Auch soll die Funktionalität hinzugefügt werden, den Game-Server über eine Kommandozeile zu starten, sodass dieser eine SEE-Instanz startet, welche als Server fungiert.

Die geplante Authentifizierung soll durch ein Passwort implementiert werden, welches beim Verbinden mit dem Server angegeben werden muss. Hierfür sollen in SEE eine Benutzeroberfläche und die notwendige Logik eingefügt werden.

Die Entwicklungsziele für den Game-Server sind also folgende:

- Ersetzen der mit *NetworkComms.Net* implementierten Logik durch *Netcode for GameObjects*
- Funktionalität für das Starten einer Server-Instanz hinzufügen
- Senden der Code-City Dateien und des aktuellen Stands der Code-City implementieren
- Starten der Server-Instanz über die Kommandozeile möglich machen
- Benutzeroberfläche für Authentifizierung implementieren
- Logik für Authentifizierung implementieren

4 Implementierung

In diesem Abschnitt möchten wir darstellen, wie wir Frontend, Backend und Game-Server implementiert haben. Hierbei wollen wir nicht nur darstellen, was wir gemacht haben, sondern auch weswegen wir etwas gemacht haben.

4.1 Frontend

Simon Leykum:

Im Folgenden möchten wir die wesentlichen Aspekte der Implementierung des Frontends darstellen.

Um einen übersichtlichen Einblick in die Implementierung des Frontends zu geben, haben wir uns dazu entschieden, die Beschreibung der Implementierung auf einige komplexere Teile zu beschränken.

4.1.1 Projektstruktur

Die Projektstruktur besteht im Wesentlichen aus fünf Ordnern. *components*, *contexts*, *types*, *util* und *views*.

Der Ordner *components* beinhaltet Components, also wiederverwendbare Code-Bausteine.

Im Ordner *contexts* befinden sich *Contexts*. Bei einem *Context* handelt es sich um ein React-Konzept. Dieses vereinfacht das Weiterreichen von Daten in einem Baum. Durch einen *Context* ist es möglich die Daten in allen Kinderknoten des Elternknotens zu verwenden, dies ist insbesondere dann interessant, wenn viele Kinderknoten diese Daten benötigen.[Con] Hierauf werden wir in den folgenden Abschnitten zu *Kommunikation zwischen Frontend und Backend* und zu dem *Anmelden* genauer eingehen.

Da wir uns bei der Entwicklung des Frontends für Typescript entschieden haben, ist es naheliegend, dass wir Datentypen definieren wollen. Hierfür haben wir den Ordner *types* erstellt, in welchem wir die für unsere Software benötigten Datentypen definiert haben.

Zuletzt möchten wir kurz auf den Ordner *views* eingehen. Rein technisch gibt es keinen Unterschied zwischen *Views* und Components. Für unsere Zwecke benutzen wir aber den Begriff *View*, um eine komplette Seite des Frontends zu definieren. Das heißt konkret also, dass Components innerhalb des Projekts wiederverwendet werden können, wobei *Views* nur einmal verwendet werden sollen.

Das Ziel der gewählten Ordnerstruktur ist es, eine klare Struktur in die Entwicklung zu bringen und für spätere Weiterentwicklung eine Einarbeitung in die Software zu erleichtern.

Außerhalb dieser Ordner befindet sich neben einigen allgemeinen Dateien der *Router* auf welchen wir nun im kommenden Abschnitt eingehen möchten.

4.1.2 Routing

Das Routing, also das Anzeigen einer bestimmten Seite, abhängig von der aktuellen URL, läuft in unserem Frontend mithilfe der Library *react-router-dom*.

Um das Routing konkret für unser Projekt zu implementieren, haben wir einen Component `Router` erstellt. Dieser zeigt abhängig davon, ob die Benutzer*in angemeldet ist oder nicht, entweder die allgemein zugänglichen Routen oder die Routen, die nur für angemeldete Benutzer*innen zugänglich sind.

```
function Router() {
  const { user } = useContext(AuthContext);

  return (
    <BrowserRouter>
      {user ? <PrivateRoutes/> : <PublicRoutes/>}
    </BrowserRouter>
  )
}
```

Wenn die Benutzer*in nicht angemeldet ist, wird ihr nur die Anmelde-Ansicht angezeigt, insofern sehen die `PublicRoutes` wie folgt aus:

```
function PublicRoutes() {
  return(
    <Routes>
        <Route path="*" element={<LoginView/>}/>
    </Routes>
  )
}
```

Dadurch, dass die Variable `path` hier auf * gesetzt ist, wird, egal welcher Pfad in der URL angegeben wird, immer die Anmelde-Ansicht angezeigt.

Im Gegensatz dazu wird bei den `PrivateRoutes` immer ein Pfad angegeben. Wenn die URL im Browser nun auf diesen Pfad zeigt, wird der dort beschriebene Component bzw. *View* angezeigt.

```
function PrivateRoutes() {
  return(
    <Routes>
        <Route path="/" element={<HomeView/>}/>
        <Route path="/server" element={<ServerView/>}/>
        <Route path="/createServer" element={<CreateServerView/>}/>
        <Route path="/settings" element={<SettingsView/>}/>
        <Route path="/personalSettings" element={<PersonalSettingsView/>}
        ↪ />
      </Routes>
  )
}
```

4.1.3 Kommunikation zwischen Frontend und Backend

In unserem Fall funktioniert die Kommunikation durch eine sogenannte *REST-Api*. Wenn hierbei Daten vom Backend angefordert werden sollen oder Daten an das Backend geschickt werden sollen werden diese mithilfe eines *Requests*, also einer Anfrage an das Backend, gesendet bzw. von dem Backend angefordert. Diese Anfrage funktioniert mithilfe einer URL, welche die Adresse des Backends angibt und einigen weiteren Informationen, die an das Backend geschickt werden. Nachdem das Backend diese Anfrage erhalten hat, führt dieses möglicherweise eine Funktion aus, um z. B. Daten aus einer Datenbank zu bekommen, und sendet dann an das Frontend eine Antwort zurück, z. B. mit den Daten aus der Datenbank.

Um diese Kommunikation auf Frontend-Seite zu implementieren, haben wir die Library *axios* verwendet.

Hierfür haben wir zunächst eine `axiosInstance` definiert. Diese erlaubt, dass immer wieder benötigte Daten, wie z. B. die URL des Backends für alle Anfragen automatisch gesetzt werden können und nicht immer wieder erneut angegeben werden müssen.

```
const axiosInstance = axios.create({
  baseURL: "http://localhost:8080/api/v1",
  timeout: 5000,
  withCredentials:true
});
```

Wie zu sehen ist, geben wir hier die `baseURL`, also die URL des Backends, ein `timeout`, nach welchem eine Anfrage an das Backend abgebrochen wird und den Parameter `withCredentials` an.

Der Parameter `withCredentials` ist notwendig, damit die Authentifizierung funktioniert. Dieser Parameter führt dazu, dass alle Cookies, die für die Webseite gespeichert sind, bei allen Anfragen mitgesendet werden.

Wenn wir nun eine Anfrage an das Backend schicken möchten, können wir dies zum Beispiel wie folgt tun:

```
axiosInstance.get(`/user/me`).then((response) => setUser(response.data));
```

Zunächst geben wir an, was wir tun möchten. Da wir in diesem Fall `axiosInstance.get(...)` ausführen, wollen wir also eine Information von dem Backend bekommen.

Für Unsere Anfrage geben wir nun die URL an, diese setzt sich aus der vorher angegebenen `baseURL`, die wir hier nicht noch einmal angeben müssen, und dem Text innerhalb der ersten Klammern zusammen. Die gesamte URL wäre für dieses Beispiel also

`http://localhost:8080/api/v1/user/me`

In diesem Beispiel warten wir nach dem Senden der Anfrage auf die Antwort und benutzen dann diese Antwort, um etwas mit den angefragten Daten zu machen.

Da die erstellte `axiosInstance` jedes Mal benutzt wird, wenn wir irgendetwas vom Backend bekommen wollen oder zum Backend schicken wollen, haben wir einen *Context* hinzugefügt. Dieser kann die `axiosInstance` an viele Components weitergeben. Auf die Funktionsweise von Contexts und unsere Implementierung des benötigten Contexts wollen wir nun im kommenden Abschnitt eingehen.

4.1.4 Context

Wenn, wie z. B. im vorherigen Abschnitt beschrieben, Variablen eines Components von vielen anderen Components abgerufen werden sollen, stellt sich die Frage, wie diese Variablen am besten an diese Components weitergegeben werden sollen.
Grundsätzlich wäre es möglich, die Variablen manuell von Component zu Component weiterzugeben, dies ist aber aufwendig und verwendet viel unnötigen Code.

Um dieses Problem zu lösen, bietet React die Möglichkeit Daten mithilfe eines *Contexts* weiterzugeben. React-Anwendungen sind als eine Art Baum aufgebaut, in der Components ineinander genistet sind. Das heißt also, es gibt immer einen Wurzelknoten, von welchem aus die ganze Applikation aufgebaut ist.
Ein Context erlaubt es nun Variablen innerhalb dieses Baumes an beliebige Kinderkomponenten, also Components, welche von einem Elternkomponenten abstammen, zu "teleportieren", sodass sie nicht von Component zu Component manuell weitergegeben werden müssen. [Con]

Um nun unseren Context zu implementieren, definieren wir zunächst ein `Interface`, eine Art Modell, welches beschreibt, wie der Context aufgebaut ist.

```
interface IAuthContext {
  user: User | null;
  setUser: (newState: User | null) => void;
  axiosInstance: AxiosInstance;
}
```

Das heißt, der Context besteht aus einem `user`, einer Funktion `setUser`, welche die `user`-Variable anpassen soll und der im vorherigen Abschnitt definierten `axiosInstance`.

Nun definieren wir das Objekt `initialValue`, dieses beschreibt den Wert, welchen die vorher definierten Variablen bei Initialisierung des `Contexts` haben sollen.

```
const initialValue = {
  user: null,
  setUser: () => {},
  axiosInstance: axiosInstance
}
```

Der `user` ist bei Initialisierung also noch nicht gesetzt, auch die `setUser`-Funktion ist zu diesem Zeitpunkt noch leer. Nur die `axiosInstance`, welche wir vorher definiert haben ist bereits gesetzt.

Um nun den Context zu erstellen, verwenden wir die React-Funktion `createContext`

```
const AuthContext = createContext<IAuthContext>(initialValue);
```

Damit dieser Context nun in unserer Anwendung benutzt werden kann, benötigen wir einen *Provider*, dieser gibt den Context an die Kinder-Components des Providers weiter

```
const AuthProvider = ({children}: {children?: ReactNode}) => {
  const [ user, setUser ] = useState<User |null>(initialValue.user);
    ↪ // State speichert die aktuelle Benutzer*in

```

```
  useEffect(() => {
    let isApiSubscribed = true;
    // Falls der Benutzername in der sessionStorage
    // gespeichert ist aber nicht geladen ist
    if(!user && isApiSubscribed && sessionStorage.getItem('username')){
      // Benutzer*in vom Backend fetchen
      axiosInstance.get(`/user/me`).then((response) => setUser(response.
       ↪  data));
    }
    return () => {
      isApiSubscribed = false;
    }
  }, [user])
   ↪  // Diese Funktion ausführen, wenn sich die user Variable ändert

  return (
    <AuthContext.Provider value={{user, setUser, axiosInstance}}>
      {children}
    </AuthContext.Provider>
  )
}
```

4.1.5 Avatar Generator

In diesem letzten Abschnitt zur Implementierung des Frontends möchten wir auf den Avatar-Generator eingehen, welcher dafür benutzt wird, die zufällig generierten Bilder für die Auseinanderhaltung der Game-Server zu generieren und darzustellen.

Für die Gestaltung der Avatare haben wir uns von den automatisch generierten Avataren der Seite *github.com* inspirieren lassen. Diese bestehen aus quadratischen Kacheln, welche entweder farbig oder weiß sind. Zudem sind die Kacheln horizontal in der Mitte gespiegelt.

Abbildung 8: Beispiel für einen Avatar auf der Seite *github.com*

Auch unsere Avatare sollen aus farbigen und weißen Kacheln bestehen und horizontal in der Mitte gespiegelt sein. Zudem sollen unsere Avatare aus 6 mal 6 Kacheln bestehen.

Um den Avatar zu generieren, müssen wir zum einen die Farbe der farbigen Kacheln generieren und zum anderen definieren, welche Kacheln farbig und welche Kacheln weiß sein sollen.
Um die Farbe der farbigen Kacheln zu generieren, haben wir die folgende Funktion definiert.

```
function getRandomColor(){
  const red = (Math.floor(Math.random() * 150) + 100).toString();
```

```
  const green = (Math.floor(Math.random() * 150) + 100).toString();
  const blue = (Math.floor(Math.random() * 150) + 100).toString();

  return `rgb(${red}, ${green}, ${blue})`;
}
```

Die funktion `getRandomColor` generiert drei Zufallswerte für die Kanäle Rot, Grün und Blau, aus welchen sich die Farbe für den Avatar zusammensetzt. Grundsätzlich können die Kanäle je einen beliebigen Wert von 0 bis 255 beinhalten. Wir begrenzen dieses aber auf einen Bereich von 100-150. Der angegebene Wert definiert, wie viel dieser Farbe in der gemischten Farbe beinhaltet werden soll, ein begrenzen der Werte sichert somit, dass das Endergebnis weder zu hell noch zu dunkel ist. Falls sonst durch Zufall für alle drei Kanäle die Zahl 255 generiert werden würde, würden die Kacheln, die eigentlich farbig sein sollten, auch weiß sein und somit nicht mehr von den Kacheln, die weiß sein sollen, zu unterscheiden sein.

Um nun zu definieren, welche Kacheln farbig sein sollen und welche Kacheln weiß sein sollen, definieren wir die folgende Funktion.

```
function getRandomSeed(){
  let avatarSeed = "";
  for(let i = 0; i < 18; i++){
      avatarSeed = avatarSeed + Math.round(Math.random()).toString();
  }
  return avatarSeed;
}
```

Da die Kacheln horizontal in der Mitte gespiegelt sein sollen, müssen wir anstatt für 36 Kacheln nur für 18 Kacheln definieren, ob diese farbig oder weiß sein sollen. Hierfür generieren wir 18 Zufallswerte zwischen 0 und 1, welche wir dann zu der nächsten Zahl runden, also bekommen wir 18 Werte, welche entweder 0 oder 1 sind. Diese speichern wir dann als eine Zeichenkette. Bei dem Anzeigen des Avatars kann diese Zeichenkette jetzt benutzt werden, um hieraus den Avatar zu bauen, wobei 0 eine Kachel beschreibt, welche weiß sein soll und 1 eine Kachel beschreibt, welche farbig sein soll.

Der Component, welcher den Avatar anzeigt, ist nun wie folgt definiert.

```
function Avatar(props: {width:number, height:number, avatarSeed: string,
 ↪  avatarColor: string}) {
    const canvasRef = useRef<HTMLCanvasElement | null>(null);

    useEffect(() => {
        const ctx = canvasRef.current?.getContext('2d');

        const widthUnit = props.width/6;
        const heightUnit = props.height/6;

        if(ctx){
            ctx.clearRect(0,0, props.width, props.height);
            ctx.fillStyle = props.avatarColor;
            ctx.globalCompositeOperation = "lighter";
             ↪  //Fix gegen weiße Ränder um Kacheln

            for (let i = 0; i < props.avatarSeed.length; i++) {
                const char = props.avatarSeed.charAt(i);
                if( char == '0'){
                    continue;
                }
                if(i % 3 == 0){
```

```
21                      ctx?.fillRect(0, heightUnit*Math.floor(i/3), widthUnit,
                         ↪   heightUnit);
22                      ctx?.fillRect(widthUnit*5, heightUnit*Math.floor(i/3),
                         ↪   widthUnit, heightUnit);
23                  } else if( (i-1) % 3 == 0){
24                      ctx?.fillRect(widthUnit, heightUnit*Math.floor(i/3),
                         ↪  widthUnit, heightUnit);
25                      ctx?.fillRect(widthUnit*4, heightUnit*Math.floor(i/3),
                         ↪   widthUnit, heightUnit);
26                  } else {
27                      ctx?.fillRect(widthUnit*2, heightUnit*Math.floor(i/3),
                         ↪   widthUnit, heightUnit);
28                      ctx?.fillRect(widthUnit*3, heightUnit*Math.floor(i/3),
                         ↪   widthUnit, heightUnit);
29                  }
30              }
31          }
32      });
33
34      return (
35          <canvas width={props.width} height={props.height} ref={canvasRef}>
36          </canvas>
37      );
38  }
```

Der Component bekommt zunächst die Breite und Höhe übergeben, die der Avatar am Ende auf der Seite ausfüllen soll. Zudem wird der `avatarSeed` und die `avatarColor` übergeben, welche zuvor von den beiden Funktionen `getRandomSeed` und `getRandomColor` generiert worden sind.

Um den Avatar zu zeichnen, verwenden wir das HTML-Element `canvas`, welches eine manipulierbare Zeichenfläche definiert. Damit wir diese manipulieren können, übergeben wir dieser einen `canvasRef`, welches eine Variable ist, die auf dieses Element zeigt.

Der Code innerhalb von `useEffect` wird nach dem Initialisieren des `canvas`-Elements ausgeführt. Hier werden zunächst die Variablen `widthUnit` und `heightUnit` definiert, welche die Größe einer Kachel innerhalb des Avatars definieren. Danach wird die Farbe der farbigen Kacheln gesetzt und Kachel für Kachel der Avatar aufgebaut.

Abbildung 9: Beispiel für einen, von uns generierten, Avatar

4.2 Backend

Thorsten Friedewold:

Dieser Abschnitt behandelt die Implementierung des Backends, hierbei wollen wir uns die Herausforderungen der Implementierung und deren Lösung genauer anschauen.

4.2.1 Technische Umsetzung

Wie bereits in der Konzeption angesprochen verwenden wir für die Implemtierung des Backends Spring Boot.

4.2.2 Struktur

Das Backend unterteilt sich im wesentlichen und drei Schichten.

- Controller: Der Controller empfängt Anfragen vom Frontend und leitet sie an den Service-Layer weiter. Darüber hinaus obliegt ihm die Verantwortung für die Autorisierung und Authentifizierung der einzelnen Anfragen aus dem Frontend.
- Service: Der Service verarbeitet die Anfragen, die er vom Controller erhält, und implementiert die eigentliche Geschäftslogik. Er greift auf die DAOs zu, um Daten zu erstellen, anzupassen oder zu löschen, und somit die gewünschten Funktionalitäten zu erreichen.
- DAO (Data Access Object): Das DAO fungiert als Schnittstelle zwischen dem Backend und dem Speicherort der Daten, in unserem Fall einer Datenbank. Es kann Daten aus der Datenbank lesen, ändern und löschen. Diese Funktionen werden an den Service weitergegeben, um die Geschäftslogik auf die Daten anzuwenden.

Dieses Verhältnis der einzelnen Schichten lässt sich in dem Folgenden Diagramm noch genauer anschauen.

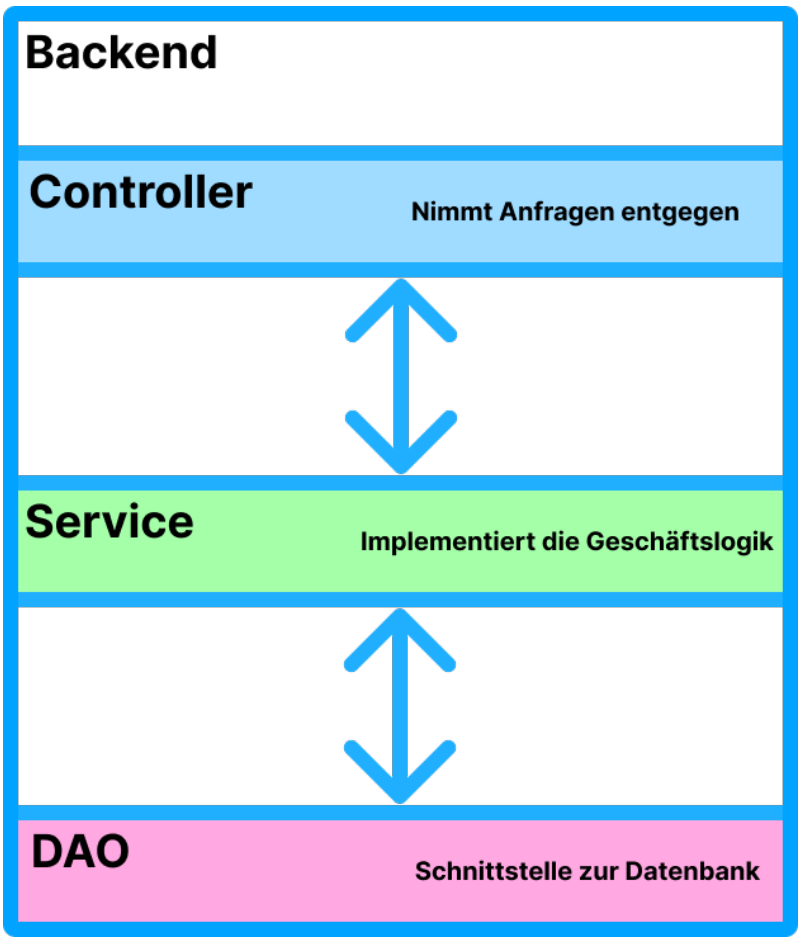

Abbildung 10: Schichten des Backends

4.2.3 Datenbankmodell

Nun wollen wir uns auf die von uns Implementierten Entitäten konzentrieren. Diese Stellen die Grundlage für unser Backend dar.

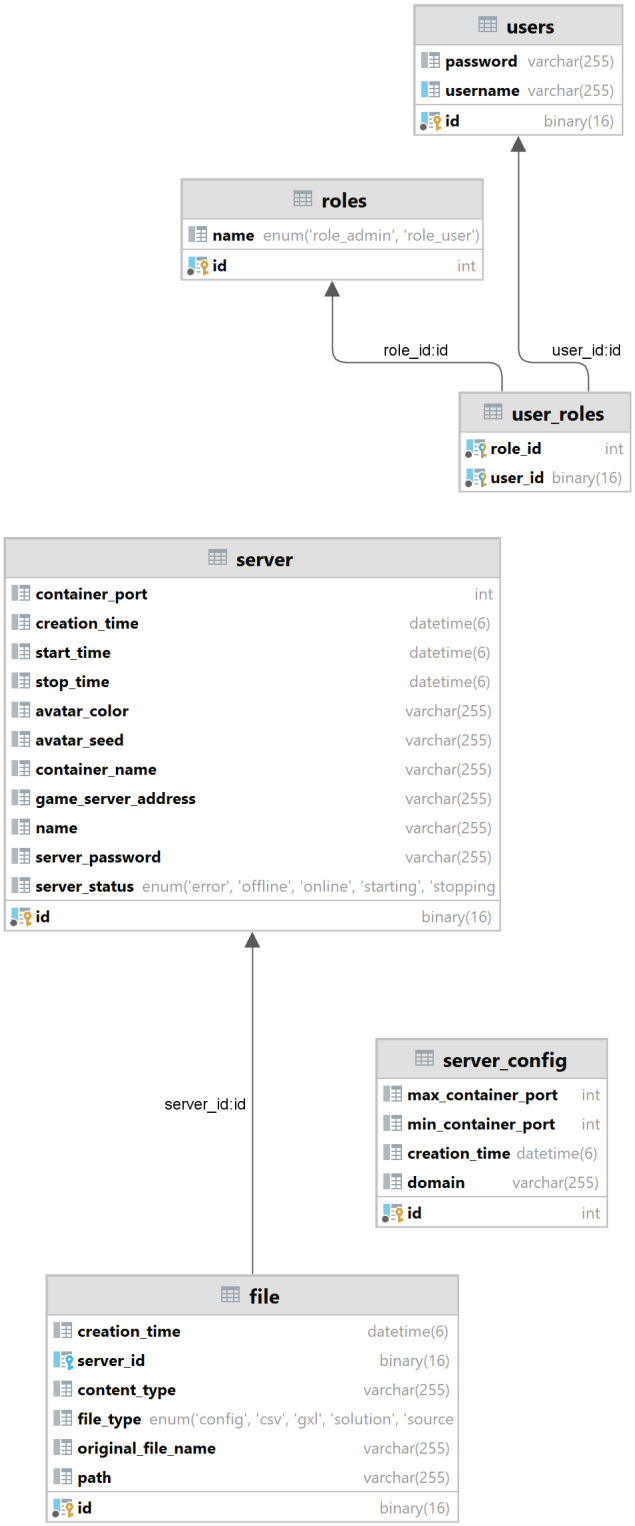

Abbildung 11: Angestrebte Architektur

Wie zu erkenne gibt es 6 verschiedene Entitäten, welche wir nutzen um alle anfallenden Daten zu speichern.

- Users: Stellt eine Nutzer*in dar.

- User-Roles: Dient zur Hilfe um eine Rolle mit einem User zu verknüpfen
- Roles: Stellt eine Rolle (für Zugriffsrechte) dar.
- Server: Stellt einen Game-Server dar.
- File: Stellt eine Datei dar.
- Server-Config: Speichert Konfigurationen für das Backend

4.2.4 Sicherheit

Damit nur Nutzer*innen die berechtigt sind Anfragen an das Backend zu senden auch eine Antwort erhalten nutzen wir in unserem Backend *JWT [Lub22]*. JWT (Json Web Token) ist ein genormtes Access-Token, dass es ermöglicht die Kommunikation zwischen zwei Teilnehmern zu sichern. Es besteht im Wesentlichen aus drei Teilen:

- *Header*. Der Header beschreibt unter anderem die verwendete Verschlüsselungmethode des Token.
- *Payload*. Der Payload beinhaltet den eigentlichen Inhalt des Tokens
- *Signatur*. Die Signatur ist eine Art Unterschrift, mittels der die Authentizität des Tokens sichergestellt wird.

Die Funktionsweise von JWT lässt sich wie folgt beschreiben.

- Eine Nutzer*in sendet Nutzernamen und Passwort an das Backend.
- Das Backend erhält die Anfrage und überprüft die gesendeten Daten.
- Wenn die Daten korrekt sind sendet das Backend einen *JWT* an die Nutzer*in zurück. Sonst nicht.
- Die Nutzer*in erhät den *JWT* und speichert ihn ab.
- Wenn die Nutzer*in erneut eine Anfrage an das Backend stellt, sendet sie nicht Nutzernamen und Passwort, sondern den gespeicherten Token mit.
- Wenn das Backend die Anfrage erhält kann es durch die Signatur erkennen, ob es den Token selbst ausgestellt hat und dementsprechend eine Antwort an die Nutzer*in senden.

Im folgenden wollen wir uns auf unsere Implementierung von JWT konzentrieren.
Zunächst erhält das Backend eine Anfrage von einer Nutzer*in mit einem *LoginRequest*, dieser beinhaltet *username* und *password*.

```
1 @PostMapping("/signin")
2 public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest
      loginRequest, HttpServletRequest request) {
3     Authentication authentication = authenticationManager.authenticate(new
          UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
          loginRequest.getPassword()));
4     SecurityContextHolder.getContext().setAuthentication(authentication);
5     UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal
          ();
6     ResponseCookie jwtCookie = jwtUtils.generateJwtCookie(userDetails);
7
8     return ResponseEntity.ok().header(HttpHeaders.SET_COOKIE, jwtCookie.
          toString())
```

```
            .body(userService.getUserByUsername(userDetails.getUsername()));
}
```

Listing 1: Anmelden

Nachdem das Backend die von der Nutzer*in übermittelten Daten überprüft hat, erstellt es mittels der folgenden Methode einen *JWT*.

```
public String generateTokenFromUsername(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date((new Date()).getTime() +
                   jwtExpirationMs))
                .signWith(key(), SignatureAlgorithm.HS256)
                .compact();
    }
}
```

Listing 2: Erstellung JWT

Wie zu erkennen setzt das Backend in den Payload des Tokens den *username*, das Ausstelldatum und das Ablaufdatum. Es ist auch zu erkennen, dass es für die Verschlüsselung ein *HS265* Algorithmus verwendet.

Nachdem das Backend den *JWT* erstellt hat, wird dieser an die Nutzer*in zurück gesendet.

4.2.5 Verwalten des Gameservers

Das Starten der einzelnen Game-Server soll vom Backend aus möglich sein. Dazu implementieren wir einige Methoden im Container Service des Backends

- Starten des Game-Servers

```
    public boolean startContainer(@NotNull Server server, List<File> files)
        {
        // Server Config laden
        Optional<ServerConfig> serverConfig = serverConfigRepo.
            findServerConfigById(1);

        // Überprüfen, ob der Server gerade beschäftigt ist
        if (serverConfig.isEmpty()) {
            log.error("Cant start server {}, cant find server config",
                server.getId());
            return false;
        }
        if (server.getServerStatusType().equals(ServerStatusType.ONLINE)
                || server.getServerStatusType().equals(ServerStatusType.
                    STARTING)
                || server.getServerStatusType().equals(ServerStatusType.
                    STOPPING)) {
            log.error("Cant start stop server {}, server is online or busy"
                , server.getId());
            return false;
        }

        // Aussuchen eines zufälligen Ports
```

```
18          int port = getRandomNumberUsingNextInt(serverConfig.get().
                getMinContainerPort(), serverConfig.get().getMaxContainerPort()
                );
19          String containerName = server.getName() + "-" + server.getId() + "-
                " + port;
20
21          // Starten des Gameservers
22          server.setServerStatusType(ServerStatusType.STARTING);
23          try {
24              log.info("Starting server {}", server.getId());
25              // Befehl zum Starten eines Docker-Containers anpassen
26              String dockerCommand = "docker run -d --name " + containerName
                    + " -p " + port + ":" + port + "/udp -e PASSWORD=\"" +
                    server.getServerPassword() + "\" -e PORT=" + port + " -e
                    SERVERID=" + server.getId() + " -e BACKENDDOMAIN=" +
                    serverConfig.get().getDomain() + " see-gameserver:latest";
27
28              // Docker-Befehl ausführen
29              Process process = new ProcessBuilder()
30                      .command("bash", "-c", dockerCommand)
31                      .inheritIO()
32                      .start();
33
34              // Warte auf den Abschluss des Prozesses
35              process.waitFor();
36
37              String mkdirCommand = "docker exec " + containerName + " mkdir
                    -p /app/gameserver_Data/StreamingAssets/Multiplayer/";
38              Process createFolderCommand = new ProcessBuilder()
39                      .command("bash", "-c", mkdirCommand)
40                      .inheritIO()
41                      .start();
42
43              // Warte auf den Abschluss des Prozesses
44              createFolderCommand.waitFor();
45
46              boolean containsZip = false;
47
48              // Kopieren der Daten in den Docker Container
49              log.info("Adding files to server {}", server.getId());
50              for (File file : files) {
51
52                  String command = "docker cp " + "\"" + file.getAbsolutePath
                        () + "\"" + " " + containerName + ":/app/
                        gameserver_Data/StreamingAssets/Multiplayer/";
53                  Process copyProcess = new ProcessBuilder()
54                          .command("bash", "-c", command)
55                          .inheritIO()
56                          .start();
57
58                  // Warte auf den Abschluss des Prozesses
59                  copyProcess.waitFor();
60
61                  if (file.getName().equals("src.zip")){
62
63                      log.error("Filename: {}", file.getName());
```

```
                    containsZip = true;
                }
            }

            // Entpacken des Zip Archives
            if (containsZip){
                String unzipCommand = "docker exec " + containerName + "
                    unzip /app/gameserver_Data/StreamingAssets/Multiplayer/
                    src.zip -d /app/gameserver_Data/StreamingAssets/
                    Multiplayer/src/";

                Process unzipProcess = new ProcessBuilder()
                        .command("bash", "-c", unzipCommand)
                        .inheritIO()
                        .start();

                // Warte auf den Abschluss des Prozesses
                unzipProcess.waitFor();
            }

            // Server in der Datenbank aktualisieren
            server.setContainerPort(port);
            server.setContainerName(containerName);
            server.setContainerAddress(serverConfig.get().getDomain());
            server.setServerStatusType(ServerStatusType.ONLINE);

            return true;

        } catch (IOException | InterruptedException e) {
            server.setServerStatusType(ServerStatusType.ERROR);
            log.error("Cant start server {}", server.getId());
        }
        return false;
    }
```

Listing 3: Starten des Game-Servers aus dem Backend

Das Backend erstellt einen neuen Container. Nach dem Starten des Containers wird ein neuer Ordner erstellt. Abschließend werden die Dateien für den Game-Server in den Container kopiert, damit dieser am Ende die Dateien den entsprechenden Clients senden kann.

- Löschen des Game-Servers

```
public boolean deleteContainer(@NotNull Server server) {
    //Kontrollieren, ob der Gameserver gerade beschäftigt ist.
    if (server.getServerStatusType().equals(ServerStatusType.OFFLINE)
        || server.getServerStatusType().equals(ServerStatusType.STARTING)
        || server.getServerStatusType().equals(ServerStatusType.STOPPING))
            {
        log.error("Cant start stop server {}, server is offline or busy",
            server.getId());
        return false;
    }
    server.setServerStatusType(ServerStatusType.STOPPING);
    try {
        log.info("Stopping server {}", server.getId());
```

```
12          // Befehl zum stoppen des Docker containers
13          String dockerStopCommand = "docker stop " + server.getContainerName
                ();
14
15          // Stoppen des Docker Containers
16          Process processStop = new ProcessBuilder()
17                  .command("bash", "-c", dockerStopCommand)
18                  .inheritIO()
19                  .start();
20
21          // Warte auf den Abschluss des Befehls
22          processStop.waitFor();
23
24          String dockerRemoveCommand = "docker rm " + server.getContainerName
                ();
25
26          // Löschen des Docker Containers
27          Process processRemove = new ProcessBuilder()
28                  .command("bash", "-c", dockerRemoveCommand)
29                  .inheritIO()
30                  .start();
31
32          // Warte auf den Abschluss des Befehls
33          processRemove.waitFor();
34
35          // Zurücksetzen des Servers in der Datenbank
36          server.setContainerPort(null);
37          server.setContainerName(null);
38          server.setContainerAddress(null);
39          server.setServerStatusType(ServerStatusType.OFFLINE);
40          return true;
41
42      } catch (IOException | InterruptedException e) {
43          server.setServerStatusType(ServerStatusType.ERROR);
44          log.error("Cant stop server {}", server.getId());
45      }
46      return false;
47  }
```

Listing 4: Löschen des Gameservers aus dem Backend

Das Backend stoppt den zu löschenden Container in dem der Game-Server läuft, danach wir der Container gelöscht und die Daten in der Datenbank zurückgesetzt.

4.2.6 Beispiel Anfrage an das Backend

Um den Ablauf einer Anfrage an das Backend besser verstehen zu können schauen wir uns anhand eines Beispiels die Schritte an, die von der Anfrage bis zur Antwort geschehen

1. Anfrage an das Backend

```
1  @GetMapping("/all")
2  @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
3  public ResponseEntity<?> getServers() {
4      return ResponseEntity.ok().body(serverService.getAllServer());
5  }
```

Listing 5: Controller

Das Backend erhält eine Anfrage und schaut sich zunächst den *JWT* an. Daraus kann es erfahren, ob der User die nötigen Rechte besitzt um die Anfrage auszuführen. In diesem Fall braucht der User die Rolle *ADMIN*, oder *USER*. Wir gehen in diesem Fall davon aus, dass der User diese Rechte besitzt. Die Anfrage wird danach an die Funktion *serverService.getAllServer()* weitergegeben.

2. Suche nach allen Servern

```
public List<Server> getAllServer() {
    log.info("Fetching all servers");
    return serverRepo.findAll();
}
```

Listing 6: Service

In *serverService.getAllServer()* gibt das Backend eine *Log*-Ausgabe aus und übergibt die Anfrage danach an *serverRepo.findAll()*.

3. Suche nach allen Servern in der Datenbank

```
List<Server> findAll();
}
```

Listing 7: DAO label

Im *serverRepo.findAll()* fragt das Backend die Datenbank nach allen Servern, die sie gespeichert hat. Sie gibt darauf hin eine Liste mit allen Servern zurück. Sobald dies Abgeschlossen ist gibt das Backend dem User, der die Anfrage gestellt hat die Liste der Server zurück. Sollten keine Server vorhanden sein, dann gibt das Backend eine leere Liste zurück.

4.3 Game-Server

Die Implementierung des Game-Servers geht direkt aus den in 3.4 definierten Entwicklungszielen heraus. Auf die Implementierung der Lösungen für diese Entwicklungsziele gehen wir nun in den folgenden Abschnitten ein.

4.3.1 Ersetzen der mit *NetworkComms.Net* implementierten Logik durch *Netcode for GameObjects*

Simon Leykum:

Wie bereits in 3.4 erwähnt, wird die externe Library *NetworkComms.Net* für das Versenden der *Player-Actions* verwendet. Diese sind atomare Aktionen, die eine Spieler*in ausführt. Ein Beispiel hierfür wäre das Erstellen oder Bewegen eines Knotens innerhalb eines Graphen.

Die Player-Actions werden innerhalb Unity als Klassen definiert. Um diese über das Netzwerk zu versenden, werden diese aber *serialisiert*.

Serialisierung in diesem Fall bedeutet das Konvertieren der Daten innerhalb einer Player-Action also z. B. Informationen darüber, welcher Knoten bewegt werden soll oder von wem die Player-Action ausgeführt worden ist, in einen *String* bzw. eine Kette von Zeichen. Aus diesem String kann dann die Player-Action

wieder rekonstruiert werden.
Diese Serialisierung ist nötig, da *NetworkComms.Net* nicht direkt das Versenden von Unity-Klassen-Instanzen unterstützt. Somit müssen diese für das Versenden mithilfe dieser Library, in ein Format konvertiert werden, welches von *NetworkComms.Net* unterstützt wird.

In der aktuellen Version funktioniert das Versenden einer *Player-Action* wie folgend:
Innerhalb von SEE wird der Ablauf mit dem Erstellen einer Instanz einer Implementierung der `AbstractNetAction` und dem Ausführen der `Execute(recipients)` Funktion dieser Instanz angestoßen, hierbei kann eine Liste von Empfängern (`recipients`) angegeben werden.
In dieser Funktion wird ein `ExecuteActionPacket` erstellt, welches die Player-Action umschließt. Dieses Paket wird mithilfe der Funktion `SubmitPacket(connection,packet)` an das `Network`-Skript weitergegeben, wo das Paket serialisiert und an eine Liste von serialisierten Paketen angefügt wird. Das `Network`-Skript beinhaltet eine Implementierung der Funktion `LateUpdate()`. Dies ist eine Unity-Funktion, welche zu jedem Frame ausgeführt wird.[Lat]
Von dieser Funktion aus werden die serialisierten Pakete von einem `PacketSequencePacket` umschlossen, dieses wird erneut serialisiert und mithilfe von *NetworkComms.Net* an den Server versendet.

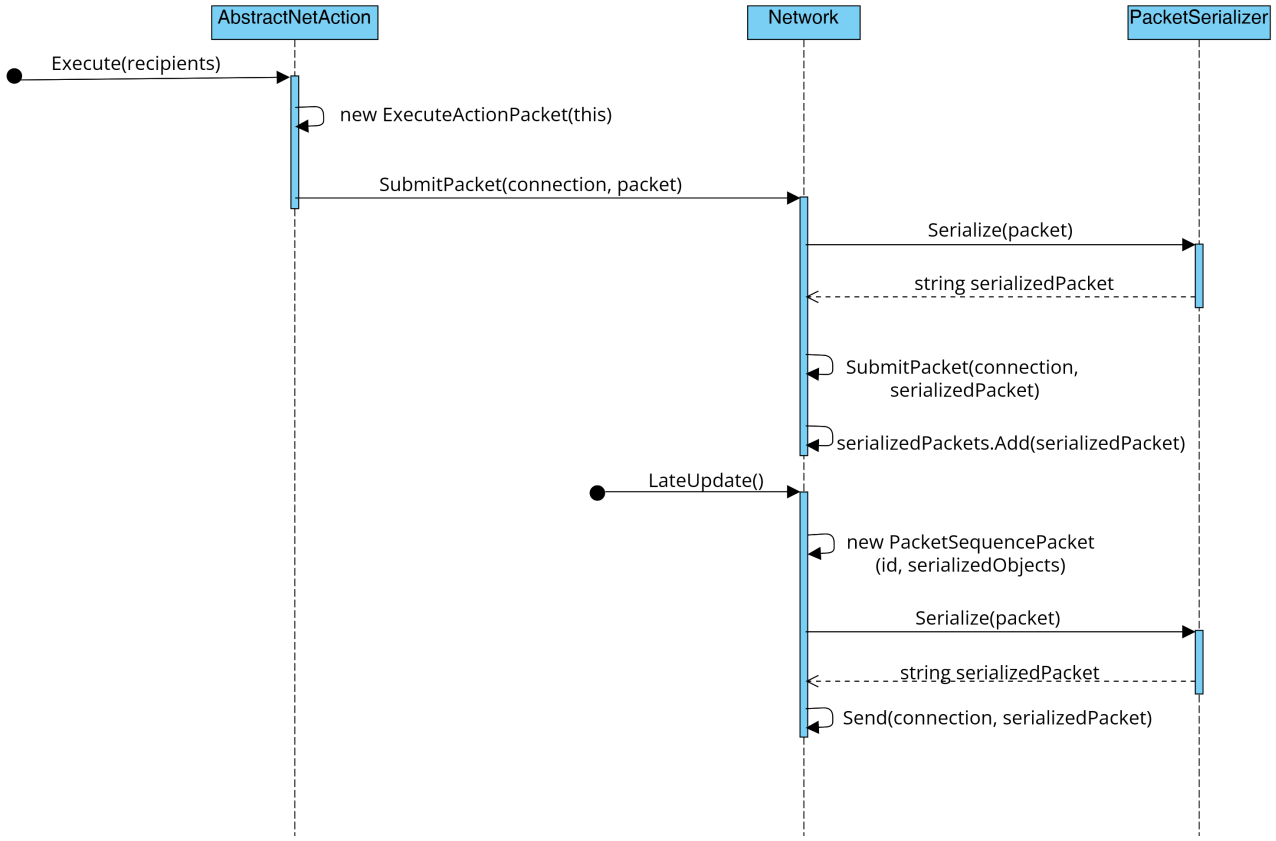

Abbildung 12: Versenden einer Player-Action nach aktueller Implementierung

Bei Initialisieren des Servers wurde ein *Callback* `OnIncomingPacket(packetHeader, connection, data)` definiert. Eine *Callback*-Funktion wird einer anderen Funktion als Parameter übergeben und bei ausführen der anderen Funktion ausgeführt. In diesem Fall wird der Callback bei dem Empfangen eines Pakets durch *NetworkComms.Net* aufgerufen. Die Callback-Funktion sendet in diesem Fall das Paket an das `PacketHandler`-Skript, hier wird aus den empfangenen Daten ein `SerializedPendingPacket`

erstellt.
Zu einem späteren Zeitpunkt wird die Implementierung der Unity-Funktion `LateUpdate()` in dem `Network`-Skript ausgeführt. Diese führt wiederum die Funktion `Update()` in dem `Server`-Skript aus. Hiernach wird mithilfe des `PacketHandler`-Skripts die Liste der `SerializedPendingPackets` durchlaufen. Nun wird zunächst aus jedem `SerializedPendingPacket` das enthaltene, serialisierte `PacketSequencePacket` deserialisiert und demnach von einem `TranslatedPendingPacket` umschlossen. Danach wird die Funktion `ExecuteOnServer(connection)` des `TranslatedPendingPackets` ausgeführt, welches wiederum die gleichnamige Funktion des beinhalteten `PacketsequencePackets` ausführt. In dieser Funktion werden zunächst alle enthaltenen `ExecuteActionPackets` deserialisiert und dann die wieder gleichnamige Funktion `ExecuteOnServer(connection)` der `ExecuteActionPackets` ausgeführt. Diese führt die `ExecuteOnServerBase()`-Funktion der `AbstractNetAction` aus, um die Player-Action auf dem Server auszuführen. Zudem wird jedes `ExecuteActionPacket` an das `Network`-Skript weitergegeben. Von wo aus es zuletzt an alle mit dem Server verbundenen Clients weitergegeben wird.

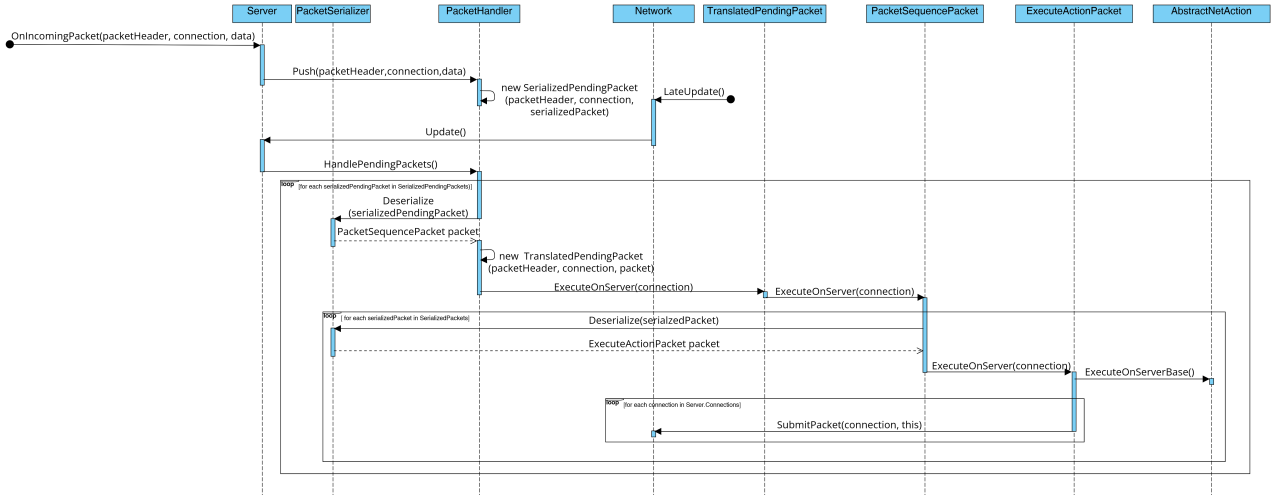

Abbildung 13: Empfangen und weitersenden der Player-Action auf Serverseite nach aktueller Implementierung

Das Empfangen und Ausführen der Player-Actions auf Client-Seite funktioniert grundsätzlich gleich wie bei dem Server mit dem Unterschied, dass die Player-Actions nicht im letzten Schritt weitergegeben werden.

Da es sich bei *NetworkComms.Net* um eine Library mit einem relativ geringen Abstraktionsgrad handelt, ist relativ viel eigene Logik für das Versenden der *Player-Action*s notwendig. So müssen unter anderem die Verbindungen selber verwaltet werden und dafür gesorgt werden, dass die versendete Pakete in der richtigen Reihenfolge bei den Empfänger*innen ankommen. Des Weiteren müssen auch die Pakete selbst definiert werden.

Durch den Wechsel zu *Netcode for GameObjects* können wir viel dieser Logik vereinfachen. Das Versenden der *Player-Action*s haben wollen wir hier durch Remote Procedure Calls (RPCs) implementiert.
RPCs erlauben es, Code auf einer fremden Instanz auszuführen.
Netcode for GameObjects bietet eine Implentierung von RPCs als Schnittstelle an, um von einem Client aus Code auf einem Server auszuführen bzw. umgekehrt.[Nfge]

In unserer Implementierung soll somit, nachdem das Ausführen einer *Player-Action* in Gang gesetzt wurde,

diese zunächst vom Client ausgeführt werden, welcher das Versenden der *Player-Action* angestoßen hat. Hiernach soll die Player-Action, nachdem sie serialisiert worden ist, über einen RPC zu dem Server gesendet werden. Dieser Server soll dann wiederum über RPCs diese *Player-Action* an die verbundenen Clients verteilen. Hier wird die *Player-Action* deserialisiert und ausgeführt.

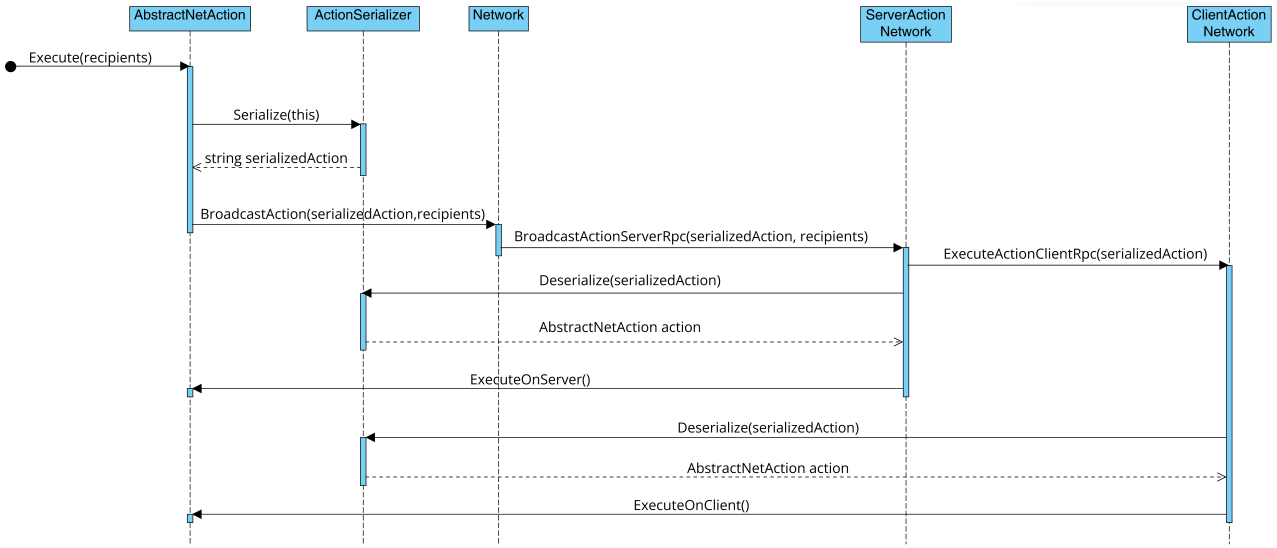

Abbildung 14: Kommunikationsverhalten nach unserer Implementierung

Um die angestrebte Funktionalität zu implementieren, haben wir zunächst zwei Unity-Skripte erstellt. Zum einen das `ClientActionNetwork`-Skript, welches die Netzwerklogik auf der Seite des Clients implementiert und zum anderen das `ClientServerNetwork`-Skript, welches die Netzwerklogik auf der Seite des Servers implementiert.
Das `ClientActionNetwork`-Skript beinhaltet die Funktion `ExecuteActionClientRpc`, welche eine übergebene serialisierte, Player-Action deserialisiert und diese auf dem Client ausführt.
Das `ClientServerNetwork`-Skript beinhaltet die Funktion `BroadcastActionServerRpc`, welche eine übergebene serialisierte Player-Action an alle bzw. alle übergebenen Clients überträgt. Diese Funktion hat zudem die Anmerkung `[ServerRpc(RequireOwnership = false)]`, diese Anmerkung ist notwendig, da diese Funktion von den Clients ausgeführt werden soll. Grundsätzlich kann nur der Client, welchem ein Netzwerk-Objekt gehört, dessen Funktionen ausführen. Durch die Anmerkung ist es aber auch für andere Clients möglich, die Funktionen auszuführen.[Nfgf][Nfgc]

```
1  public class ServerActionNetwork : NetworkBehaviour
2  {
3      //[...]
4
5      [ServerRpc(RequireOwnership = false)]
6      public void BroadcastActionServerRpc(string serializedAction, ulong[]
        ↪  recipients)
7      {
8          if (!IsServer && !IsHost)
9          {
10             return;
11         }
12
```

```
13          AbstractNetAction deserializedAction = ActionSerializer.
             ↪  Deserialize(serializedAction);
14          if (deserializedAction.ShouldBeSentToNewClient)
15          {
16              Network.NetworkActionList.Add(serializedAction);
17          }
18          deserializedAction.ExecuteOnServer();
19          foreach (NetworkClient client in NetworkManager.Singleton.
             ↪  ConnectedClientsList)
20          {
21              if(recipients == null || recipients.Contains(client.ClientId))
                 ↪   {
22                  ClientActionNetwork clientNetwork = client.PlayerObject.
                     ↪  GetComponent<ClientActionNetwork>();
23                  clientNetwork.ExecuteActionClientRpc(serializedAction);
24              }
25          }
26      }
27  }
28
29  public class ClientActionNetwork : NetworkBehaviour
30  {
31      //[...]
32
33      [ClientRpc]
34      public void ExecuteActionClientRpc(string serializedAction)
35      {
36          if (IsHost  || IsServer)
37          {
38              return;
39          }
40          AbstractNetAction action = ActionSerializer.Deserialize(
             ↪  serializedAction);
41          if(action.Requester != NetworkManager.Singleton.LocalClientId)
42          {
43              action.ExecuteOnClient();
44          }
45      }
46  }
```

Beide implementierte Skripte sind Implementierungen der `NetworkBehaviour`-Klasse, diese Klasse ist Teil der *Netcode for GameObjects* Library und erlaubt unter anderem das Verwenden von RPCs.[Nfgb].

Skripte, die von der `NetworkBehaviour`-Klasse erben müssen immer an ein GameObject angefügt werden.[Nfgd]
Das `ClientServerNetwork`-Skript haben wir an die Charaktere der Spieler*innen angefügt und für das `ClientServerNetwork`-Skript haben wir ein seperates, leeres GameObject erstellt.

Zuletzt haben wir das `Network`-Skript und die relevanten Player-Actions so angepasst, dass sie mit der neuen Implementierung funktionieren.

4.3.2 Funktionalität für das Starten einer Server-Instanz hinzufügen

Simon Leykum:

Um das Starten einer Sever-Instanz zu implementieren waren nur relativ wenige Änderungen notwendig, da *Netcode for GameObjects* bereits die Funktionalität enthält, einen Client, Server oder Host zu starten. Diese Implementierung wurde auch bisher benutzt, um den Client und Host zu starten.

Neben den kleinen Anpassungen, die notwendig waren, um die aktuelle Funktionalität zu erweitern, musste das `PlayerSpawner`-Skript angepasst werden, sodass für die Server Instanz keinen Spielcharakter in der Welt erstellt wird.

4.3.3 Senden des aktuellen Stands der Code-City implementieren

Simon Leykum:

Um das Synchronisieren des aktuellen Zustands der Code-City zu implementieren, sind wir wie folgt vorgegangen.

Zunächst haben wir in dem `Network`-Skript eine Variable hinzugefügt, welche eine Liste von serialisierten Player-Actions speichert.

```
public static List<string> NetworkActionList = new();
```

Bei dem Senden einer Player-Action mithilfe BroadcastActionSeverRpc wird diese Liste gefüllt.

```
[ServerRpc(RequireOwnership = false)]
public void BroadcastActionServerRpc(string serializedAction, ulong[]
    recipients)
{
    if (!IsServer && !IsHost)
    {
        return;
    }

    AbstractNetAction deserializedAction = ActionSerializer.Deserialize(
        serializedAction);
    if (deserializedAction.ShouldBeSentToNewClient)
    {
        //Add serialized Action to list to be synced with new clients
        Network.NetworkActionList.Add(serializedAction);
    }
    deserializedAction.ExecuteOnServer();
    foreach (NetworkClient client in NetworkManager.Singleton.
        ConnectedClientsList)
    {
        if(recipients == null || recipients.Contains(client.ClientId)) {
            ClientActionNetwork clientNetwork = client.PlayerObject.
                GetComponent<ClientActionNetwork>();
            clientNetwork.ExecuteActionClientRpc(serializedAction);
        }
    }
}
```

Damit nicht alle Player-Actions Synchronisiert werden, haben wir zudem die Property `ShouldBeSentToNewClient` zu der `AbstractNetAction` hinzugefügt.

```
public virtual bool ShouldBeSentToNewClient { get => true; }
```

Diese ist standardmäßig auf `true` gesetzt, sodass die Player-Action synchronisiert wird, kann aber von Player-Actions, die von der `AbstractNetAction`erben, überschrieben werden. Dies ist insbesondere bei Player-Actions wie die `SoundEffectNetAction` interessant, diese ist für das Synchronisieren von Soundeffekten zuständig, führt aber beim Verbinden von neuen Clients dazu, dass alle bisherigen Soundeffekte auf einmal abgespielt werden.

```
public override bool ShouldBeSentToNewClient { get => false; }
```

Auf der Seite des Clients haben wir nun die Funktion `SyncClientServerRpc(ulong client)` implementiert, diese führt auf Server-Seite die Funktion `SyncClientServerRpc` aus. Innerhalb dieser Funktion werden die gespeicherten Player-Actions durchlaufen und mithilfe `ExecuteActionUnsafeClientRpc` an den Client gesendet.

```
[ServerRpc(RequireOwnership = false)]
public void SyncClientServerRpc(ulong client)
{
    NetworkClient networkClient = NetworkManager.Singleton.
     ↪  ConnectedClientsList.FirstOrDefault((connectedClient) =>
     ↪  connectedClient.ClientId == client);
    if (networkClient == null)
    {
        return;
    }
    if(!networkClient.PlayerObject.TryGetComponent<ClientActionNetwork>(
     ↪  out var clientNetwork))
    {
        return;
    }
    foreach (string serializedAction in Network.NetworkActionList.ToList
     ↪  ())
    {
        clientNetwork.ExecuteActionUnsafeClientRpc(serializedAction);
    }
}
```

Die Funktion `ExecuteActionUnsafeClientRpc` führt die Player-Action aus, auch wenn der Client der Absender ist, da ein Client, der sich vom Server trennt und nachträglich wieder verbindet, auch die eigenen Aktionen erneut ausführen sollte.

```
[ClientRpc]
public void ExecuteActionUnsafeClientRpc(string serializedAction)
{
    if (IsHost || IsServer)
    {
        return;
    }
    AbstractNetAction action = ActionSerializer.Deserialize(
     ↪  serializedAction);
    action.ExecuteOnClient();
}
```

4.3.4 Senden der Code-City Dateien implementieren

Simon Leykum:

Um das Senden der Code-City Dateien zu implementieren, hatten wir zunächst vor, RPCs zu verwenden. Dies hat sich leider aufgrund einer Beschränkung der maximal versendbaren Daten mithilfe eines RPCs herausgestellt nicht möglich zu sein.

Unserer alternativer Ansatz war es, die Dateien mithilfe einer Anfrage an das Backend zu implementieren. So haben wir dies zuletzt auch implementiert.

Hierfür haben wir die Variable `ServerId` und `BackendDomain` in dem Skript `Network` hinzugefügt. Diese Variablen können beim Erstellen des Servers mithilfe der Parameter `-id` und `-domain` übergeben werden.

Zudem haben wir die Funktionen `SyncFilesServerRpc` und `SyncFilesClientRpc` hinzugefügt. Hierdurch kann der Client die aktuelle `ServerId` des Game-Servers bekommen. Hiernach werden die Funktionen `GetSource()`, `GetGxl()`, `GetCsv()`, `GetConfig()` und `GetSolution()` ausgeführt. Diese sind dafür zuständig, dass die Dateien von dem Backend angefragt werden. Da die Funktionen inhaltlich sehr ähnlich sind, wollen wir uns hier beispielhaft die Funktion `GetSource()` anschauen.

```
IEnumerator GetSource()
{
    using UnityWebRequest webRequest = UnityWebRequest.Get("http://"  +
     ↪  Network.BackendDomain + "/api/v1/file/client/source?serverId=" +
     ↪  Network.ServerId + "&roomPassword=" + Network.Instance.
     ↪  RoomPassword);
    webRequest.downloadHandler = new DownloadHandlerFile(Application.
     ↪  streamingAssetsPath + "/Multiplayer/src.zip");

    // Request and wait for the desired page.
    yield return webRequest.SendWebRequest();

    if (webRequest.result != UnityWebRequest.Result.Success)
    {
        Debug.LogError("Error fetching source from backend: " + webRequest.
         ↪  error);
    }
    else
    {
        try
        {
            // unzip the source code
            ZipFile.ExtractToDirectory(Application.streamingAssetsPath +
             ↪  "/Multiplayer/src.zip", Application.streamingAssetsPath +
             ↪  "/Multiplayer/src");
        }
        catch (Exception e)
        {
            Debug.LogError("Error unzipping source code: " + e.Message);
        }
    }
    //[...]
}
```

Hier wird zunächst ein `UnityWebRequest` definiert, wobei die URL, inklusive der Adresse des Backends, der `ServerId` und des Raumpassworts übergeben werden.

Danach definieren wir einen `DownloadHandler`, welcher dafür zuständig ist, die angefragte Datei zu speichern. Hierbei geben wir den Pfad inklusive des Dateinamens an.

Nun senden wir die Anfrage mithilfe von `webRequest.SendWebRequest()`.

Falls diese Anfrage nun erfolgreich war, speichert der `DownloadHandler` die Datei automatisch in dem angegebenen Pfad.

Da wir bei der Funktion `GetSource()` ein *ZIP-Archiv* erwarten, versuchen wir nach Empfangen der Datei nun diese zu entpacken.

Nun möchten wir noch auf die Implementierung der Funktion auf Seite des Backends eingehen.

Thorsten Friedewold:

Sobald ein Client eine Anfrage an das Backend sendet, um die Dateien zu erhalten, muss es dem Backend mitteilen, um welchen *Gameserver* es sich handelt, damit die richtigen Dateien gesendet werden. Im Backend wurde für jede Dateiart, die gespeichert werden kann (`GXL`, `CVS`, `CONFIG`, `SOLUTION`, `SOURCE`), ein eigener Request erstellt, damit die Clients mit möglichst wenigen Parametern die entsprechenden Dateien erhalten können.

Beispielhaft betrachten wir den Request zum Erhalten der *CONFIG*:

```
@GetMapping("/client/config")
public ResponseEntity<?> getConfig(@RequestParam("serverId") UUID serverID,
    @RequestParam("roomPassword") String roomPassword) {
// Laden des Servers aus der Datenbank
Server server = serverService.getServerByID(serverID);


\begin{lstlisting}[style=javaStyle, caption={Anmelden}, label=java-code]
    @GetMapping("/client/config")
    public ResponseEntity<?> getConfig(@RequestParam("serverId") UUID serverID,
        @RequestParam("roomPassword") String roomPassword) {
        //Laden des Servers aus der Datenbank
        Server server = serverService.getServerByID(serverID);

        // Überprüfen, ob der Request mit dem richtigen Passwort gesendet wurde
            , und ob der server existiert
        if (evalNot(server, roomPassword)) return ResponseEntity.badRequest().
            build();
        try {

            //Laden der Datei aus der Datenbank und Dateispeicher
            PayloadFile payloadFile = fileService.getFileByServerAndFileType(
                server, FileType.CONFIG);

            // Sendern der Datei an den Client
            byte[] data = payloadFile.getContent();
            ByteArrayResource resource = new ByteArrayResource(data);
            return ResponseEntity
                    .ok()
                    .contentLength(data.length)
                    .header("Content-type", payloadFile.getContentType())
                    .header("Content-disposition", "attachment; filename=\"" +
                        payloadFile.getOriginalFileName() + "\"")
                    .body(resource);

        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }
```

Listing 8: Anmelden

Wie ersichtlich, verzichten wir bei diesem Request auf *JWT* und überprüfen lediglich das entsprechende Raumpasswort.

4.3.5 Starten der Server-Instanz über die Kommandozeile möglich machen

Simon Leykum:

Das Starten einer Unity-Instanz über die Kommandozeile ist bereits grundsätzlich möglich.[Uni]

Um direkt eine Server-Instanz zu starten, müssen wir aber zusätzlich etwas eigene Logik implementieren. Hierfür haben wir die `Start()`-Funktion, des `Network`-Skripts erweitert, sodass die mitgegebenen Kommandozeilenargumente, also die Argumente, welche beim Starten von SEE mithilfe der Kommandozeile hinzugegeben werden können, durchlaufen werden und falls das Argument `-launch-as-server` gefunden wird, direkt ein Server gestartet wird, ohne das Startmenü anzuzeigen.

```
1  private void Start()
2  {
3      //[...]
4
5      //Check command line arguments
6      for (int i = 0; i < arguments.Length; i++)
7      {
8          //[...]
9
10         if (arguments[i] == "-launch-as-server")
11         {
12             StartServer(null);
13         }
14     }
15 }
```

Da für die Server-Instanz keine grafische Benutzeroberfläche existiert, soll auch nichts angezeigt werden. Hierfür kann beim Starten der Server-Instanz das Kommandozeilenargument `-batchmode` verwenden.[Uni]

Um nun eine Server-Instanz zu starten, kann das folgende Kommando verwendet werden:

```
./SEE.exe -launch-as-server -batchmode
```

4.3.6 Authentifizierung implementieren

Thorsten Friedewold:

Damit die einzelnen Server nur von ausgewählten Clients verwendert werden können, haben wir uns dazu entschlossen diese mit einem Passort zu schützen. Dieses Passwort kann beim erstellen des Gameservers gesetzt werden. Das Passwort wird wie in 4.3.5 an den Game-Server übergeben. In den Einstellungen der Clients kann ebenfalls ein Passwort gesetzt werden, dass die Clients beim verbinden an den Server senden, dieser Entscheidet dann ob eine Verbindung aufgebaut werden soll.

```
private void ApprovalCheck(NetworkManager.ConnectionApprovalRequest
    request, NetworkManager.ConnectionApprovalResponse response)
{
    if (RoomPassword == System.Text.Encoding.ASCII.GetString(request.
        Payload))
    {
        Debug.Log(
            $"Client {request.ClientNetworkId} has send right room password"
            );
        response.Approved = true;

    }
    else
    {
        response.Approved = false;
        response.Reason = "Invalid password";
        Debug.LogWarning(
            $"Client {request.ClientNetworkId} has send wrong room password"
            );
    }
}
```

Mittels dieser Methode kann der Server das von dem Client übermittelte Passwort überprüfen.

Folgend müssen wir noch dafür sorgen, dass der Client das Passwort beim Verbinden an den Server sendet. Hierfür können wir dem Client sagen, dass er beim Verbindungsaufbau das Passwort Encodiert absenden soll

```
NetworkManager.Singleton.NetworkConfig.ConnectionData = Encoding.ASCII.
    GetBytes(RoomPassword);
```

Abschließend müssen wir nur noch dafür sorgen, dass der Server das Passwort überprüft, dafür können wir folgende Zeile beim starten des Servers hinzufügen

```
NetworkManager.Singleton.ConnectionApprovalCallback = ApprovalCheck;
```

Somit werden alle Verbingungsversuche erst zugelassen, wenn das richtige Passwort eingegeben wurde.

5 Deployment

Thorsten Friedewold:

Um alle Dienste problemlos betreiben zu können, haben wir uns entschieden, diese in *Docker-Containern* auszuführen. Insbesondere beim Game-Server bietet dies den Vorteil, ihn bequem vom *Backend* aus starten und beenden zu können.

Zunächst wollen wir jedoch genauer beleuchten, was *Docker* ist.

5.1 Docker

Docker ist eine Software, mit der wir Programme innerhalb von sogenannten *Containern* ausführen können. Container sind dabei kleine, isolierte eigenständige Computer, die alle erforderlichen Ressourcen zum Starten erhalten. Wenn wir ein Programm in einem *Container* ausführen möchten, müssen wir zunächst ein *Image* erstellen. Hierfür erstellen wir zunächst ein *Dockerfile*, in dem wir festlegen, welche Komponenten das auszuführende Programm letztendlich benötigt. Nachdem wir das *Dockerfile* erstellt haben, können wir das Image generieren. Dieses können wir als eine Art Bauplan für unseren *Container* betrachten. Aus dem *Image* können wir beliebig viele *Container* erstellen.

Im Folgenden wollen wir betrachten, wie die einzelnen Dienste in *Docker-Container* überführt wurden. Wir beginnen mit der Erstellung der jeweiligen *Dockerfiles*.

5.2 Backend

Das *Backend* wird mit dem folgenden *Dockerfile* gebaut:

```
# Verwendung von einem Maven Image für das Bauen
FROM maven:3.8.4-openjdk-17-slim AS build

# Kopieren der Quellcode-Dateien und der POM-Datei
WORKDIR /app
COPY src ./src
COPY pom.xml .

# Bauen des Backends
RUN mvn clean package -DskipTests

# Verwendung von Ubuntu 20.04 für das Ausführen des Backends
FROM ubuntu:20.04

# Installieren von Java und Docker (Docker wird installiert, um den \
   gls{gameserver} zu starten)
RUN apt-get update && \
    apt-get install -y openjdk-17-jdk && \
    apt-get install docker.io -y

# Setzen von Systemvariablen für Java
ENV JAVA_HOME /usr/lib/jvm/java-17-openjdk-amd64
ENV PATH $PATH:$JAVA_HOME/bin

# Öffnen des Ports 8080
EXPOSE 8080

# Kopieren des gebauten Backends
COPY --from=build app/target/see-backend.jar see-backend.jar

# Starten des Backends
ENTRYPOINT ["java","-jar","/see-backend.jar"]
```

5.3 Frontend

Auch beim *Frontend* gehen wir ähnlich vor:

```
# Verwenden des Node 20 Base Images
FROM node:20-slim AS base

# Setzen von Systemvariablen für das Kompilieren
ENV PNPM_HOME="/pnpm"
ENV PATH="$PNPM_HOME:$PATH"
RUN corepack enable
```

```
# Kopieren des Projekts in den Docker Container
COPY . /app
WORKDIR /app

# Installieren der benötigten Module für das Frontend
FROM base AS prod-deps
RUN --mount=type=cache,id=pnpm,target=/pnpm/store pnpm install --prod --
    frozen-lockfile

# Bauen des Frontends
FROM base AS build
RUN --mount=type=cache,id=pnpm,target=/pnpm/store pnpm install --frozen-
    lockfile
RUN pnpm install
RUN pnpm run build

# Verwendung von Nginx als Webserver
FROM nginx:alpine

# Kopieren des gebauten Frontends in den Webserver
WORKDIR /usr/share/nginx/html
COPY --from=build /app/dist .

# Befehl zum Starten des Webservers
CMD ["nginx", "-g", "daemon off;"]
```

5.4 Gameserver

Leider war es nicht möglich, den Game-Server vollständig mithilfe eines *Dockerfiles* zu bauen. Daher bauen wir das Projekt in Unity und kopieren das Programm anschließend in ein *Docker-Image*.

```
# Verwendung von Ubuntu 20.04 x86/64
FROM --platform=linux/amd64 ubuntu:20.04

# Kopieren der Daten in den Container
WORKDIR /app
COPY . .

# Setzen des Raum-Passworts und des Ports, sowie der ServerID und der
    Adresse für das Backend
ENV PASSWORD=
ENV PORT=7777
ENV SERVERID=
ENV BACKENDDOMAIN=localhost:8080

# Öffnen des Ports, damit sich Clients verbinden können
EXPOSE ${PORT}

# Starten des Gameservers
```

```
CMD ./gameserver.x86_64 -port ${PORT} -password ${PASSWORD} -launch-as-
    server -batchmode
```

5.5 Ausführen

Da wir nun alle *Images* erstellen konnten, wollen wir diese im folgenden Schritt ausführen. Dafür nutzen wir eine weitere Möglichkeit von *Docker*. Wir können all unsere Container in einem sogenannten *Docker-Compose-File* festhalten und diese dann einfach mit einem Befehl zusammen starten. Für unsere Anwendung haben wir das folgende *Docker-Compose-File* erstellt:

```
version: "3.8"

# Wir erstellen ein Netzwerk, damit die Container untereinander
   kommunizieren können.
networks:
  server-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.18.0.0/16

services:
  # Frontend
  frontend-server:
    # Hier geben wir den Pfad zu unserem Frontend an, in dem das
       Dockerfile liegt, damit es gebaut werden kann.
    build: ./bachelorarbeit-frontend
    container_name: frontend
    # Wir geben an, dass das Frontend auf das Backend angewiesen ist.
    depends_on:
      - backend-server
    # Wir fügen das Frontend in das erstellte Netzwerk hinzu.
    networks:
      server-network:
        ipv4_address: 172.18.0.6
    # Wir geben an, dass das Frontend auf dem Port 80 hört.
    ports:
      - "80:80"

  # Backend
  backend-server:
    # Privileged müssen wir setzen, damit das Backend weitere Gameserver
       starten kann.
    privileged: true
    build: ./bachelorarbeit-backend
    container_name: backend
    restart: always
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
```

```
    - database-server
    - storage-server
  networks:
    server-network:
      ipv4_address: 172.18.0.2
  ports:
    - "8080:8080"
  # Hier geben wir Variablen an das Backend an. Das machen wir, damit
     wir sie beim Starten anpassen können und das Backend nicht bei
     jeder Änderung neu bauen müssen.
  environment:
    MYSQL_HOST: 172.18.0.3  # IP-Adresse des MySQL-Servers
    MYSQL_PORT: 3306
    MYSQL_DATABASE: app_db
    MYSQL_USER: db_user
    MYSQL_PASSWORD: db_user_pass
    MINIO_URL: 172.18.0.4  # IP-Adresse des Minio-Servers
    MINIO_PORT: 9000
    MINIO_DEFAULT_BUCKETS: test
    MINIO_ROOT_PASSWORD: your_password
    MINIO_ROOT_USER: your_username
    DOMAIN: localhost
    DOMAIN_PORT: 80
    JWT_SECRET: 1RNG9ja6MtadMuLsRC5ci3fy53KcQJmdEWWrp3hV5dnukvW5SQ

# Datenbank: Hier nutzen wir einen offiziellen MySQL-Server.
database-server:
  image: mysql:latest
  container_name: database
  networks:
    server-network:
      ipv4_address: 172.18.0.3
  environment:
    MYSQL_ROOT_PASSWORD: my_secret_password
    MYSQL_DATABASE: app_db
    MYSQL_USER: db_user
    MYSQL_PASSWORD: db_user_pass

# Dateispeicher: Hier nutzen wir einen offiziellen MinIO-Server.
storage-server:
  image: docker.io/bitnami/minio:latest
  container_name: storage
  networks:
    server-network:
      ipv4_address: 172.18.0.4
  environment:
    MINIO_ROOT_USER: your_username
    MINIO_ROOT_PASSWORD: your_password
    MINIO_DEFAULT_BUCKETS: test
```

Abschließend können wir alle Komponenten mit folgendem Befehl starten:

```
docker-compose up --build
```

6 Testen

Im Folgenden möchten wir darstellen, wie wir unser Frontend, Backend und den Game-Server evaluiert haben.

6.1 Frontend

Simon Leykum:

Um das Frontend zu evaluieren, haben wir eine Benutzbarkeitsstudie durchgeführt. Hierfür haben wir einige Szenarien definiert, welche die Proband*innen durchlaufen sollen.

6.1.1 Forschungsfrage

Grundsätzlich ist das Ziel der Studie, die Qualität des Frontends zu evaluieren.

Zunächst ist es uns wichtig, die Benutzbarkeit des Frontends zu analysieren. Zudem möchten wir herausfinden, ob alle Funktionen wie geplant funktionieren, auch auf diversen Browsern und Endgeräten. Zuletzt möchten wir herausfinden, ob das gewählte Design für die Probant*innen visuell ansprechend ist.

6.1.2 Methodik

Um die Korrektheit des Frontends zu analysieren, befragen wir die Probant*innen, nach dem Durchgehen jedes Szenarios, ob alles funktioniert hat und wie sicher sie in ihrer Entscheidung sind.
Die Benutzbarkeit analysieren wir mithilfe der System Usability Scale (SUS), hierbei werden 10 standardisierte Fragen gestellt. Ein wesentlicher Vorteil der SUS ist, dass diese bereits bei einer relativ kleinen Anzahl von Probant*innen eine signifikante Aussage über die Benutzbarkeit eines Systems geben kann. [Sau13]
Um herauszufinden, ob das gewählte Design den Probant*innen gefallen hat, fragen wir auch dies innerhalb der Benutzerstudie ab.

6.1.3 Fragen

Um mehr über unsere Probant*innen zu erfahren, fragen wir zunächst einige allgemeine Fragen.

- Wie alt bist du? (<18/18-20/20-25/25-30/30-35/35-40/40+)
- Was ist dein höchster Schulabschluss (Kein Abschluss/Hauptschulabschluss/Mittlerer Abschluss/Fachhoschulreife oder Allgemeine Hochschulreife/Bachelor/Master oder Höher/Anderer)
- Falls du arbeitest: was ist dein Beruf?
- Falls du studierst: was ist dein Studiengang?
- Falls bekannt: Welches Betriebssystem und welchen Browser verwendest du um diese Studie zu bearbeiten?
- Wie würdest du deine Erfahrungen im Umgang mit Software bewerten? (Skala von 1-5, 1: Sehr unerfahren, 5: Sehr erfahren)

Hiernach gehen die Probant*innen die folgenden Szenarien durch und beantworten die diesbezüglichen Fragen.

Szenario 1: Anmelden und Übersichtsseite
Voraussetzungen: • Benutzer*in ist nicht angemeldet
Ziele • Benutzer*in möchte sich anmelden • Benutzer*in möchte sich einen Überblick über die existierenden Game-Server beschaffen
Fragen • Hat alles funktioniert? (ja/nein/unsicher) • Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher) • Falls nein, was hat nicht funktioniert?

Szenario 2: Gameserver Verwaltung

Voraussetzungen:

- Benutzer*in ist angemeldet
- Benutzer*in befindet sich auf Übersichtsansicht

Ziele

- Benutzer*in möchte die Detailansicht eines Game-Servers anschauen
- Benutzer*in möchte auf dieser Seite folgende Aktionen, in angegebener Reihenfolge, durchführen:
  - Einen Server Stoppen und danach wieder Starten
  - Hochgeladene Projektdateien herunterladen
  - Die IP des Servers teilen
  - Zurück zu der Übersichtsseite gelangen

Fragen

- Hat alles funktioniert? (ja/nein/unsicher)
- Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher)
- Falls nein, was hat nicht funktioniert?

Szenario 3: Gameserver Löschen

Voraussetzungen:

- Benutzer*in ist angemeldet
- Benutzer*in befindet sich auf Übersichtsansicht

Ziele

- Benutzer*in möchte einen Game-Server löschen

Fragen

- Hat alles funktioniert? (ja/nein/unsicher)
- Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher)
- Falls nein, was hat nicht funktioniert?

Szenario 4: Gameserver Erstellen

Voraussetzungen:

- Benutzer*in ist angemeldet
- Benutzer*in befindet sich auf Übersichtsansicht
- Benutzer*in hat Projektdateien zum Hochladen

Ziele

- Benutzer*in möchte einen Game-Server erstellen, wobei zu beachten ist:
  - Benutzer*in möchte das generierte Bild für den Server neu generieren lassen

Fragen

- Hat alles funktioniert? (ja/nein/unsicher)
- Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher)
- Falls nein, was hat nicht funktioniert?

Szenario 5: Benutzer*innenverwaltung

Voraussetzungen:

- Benutzer*in ist angemeldet
- Benutzer*in ist Administrator*in

Ziele

- Benutzer*in möchte die Benutzer*innen der Organisation verwalten und dabei die folgenden Aktionen ausführen:
  - Eine neue Benutzer*in mit einem zufällig generierten Passwort hinzufügen
  - Eine Benutzer*in zur Administrator*in ernennen
  - Eine Benutzer*in aus der Organisation entfernen

Fragen

- Hat alles funktioniert? (ja/nein/unsicher)
- Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher)
- Falls nein, was hat nicht funktioniert und falls bekannt, weswegen?

Szenario 6: Benutzereinstellungen

Voraussetzungen:

- Benutzer*in ist angemeldet

Ziele

- Benutzer*in möchte den eigenen Benutzernamen und das Eigene Passwort ändern

Fragen

- Hat alles funktioniert? (ja/nein/unsicher)
- Falls ja/unsicher, wie sicher bist du dir, dass alles Funktioniert hat? (Skala von 1-5, 1: Sehr unsicher, 5: Sehr sicher)
- Falls nein, was hat nicht funktioniert?

Im Anschluss an das Durchgehen der Szenarien fragen wir zunächst ab, wie die Probant*innen die Software

visuell fanden.

- Ich fand das Tool visuell ansprechend (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)

Zuletzt fragen wir einige Fragen, um die Benutzbarkeit anhand der *System Usability Scale* zu analysieren. Die Fragen sind aus dem orginalen Fragebogen[Bro95] entnommen und ins Deutsche übersetzt worden.

- Ich kann mir vorstellen das Tool oft zu verwenden. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand das Tool unnötig komplex. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand das Tool einfach zu benutzen. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich denke, dass ich technische Hilfe benötige um das Tool zu benutzen. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand die Funktionen des Tools gut integriert. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand, dass es zu viele Ungleicheiten in dem Tool gab. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich könnte mir vorstellen, dass die meisten Menschen das Tool schnell zu bedienen lernen können. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand das Tool sehr anstrengend zu benutzen. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich fand mich bei bedienen des Tools sehr sicher. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)
- Ich musste viel lernen bevor ich das Tool benutzen konnte. (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)

6.1.4 Ergebnisse

Wir konnten für unsere Studie 11 Probant*innen finden. Unsere Probant*innen waren zum Großteil zwischen 18 und 24, die älteste Person, die an der Studie teilgenommen hat war zwischen 35 und 40.

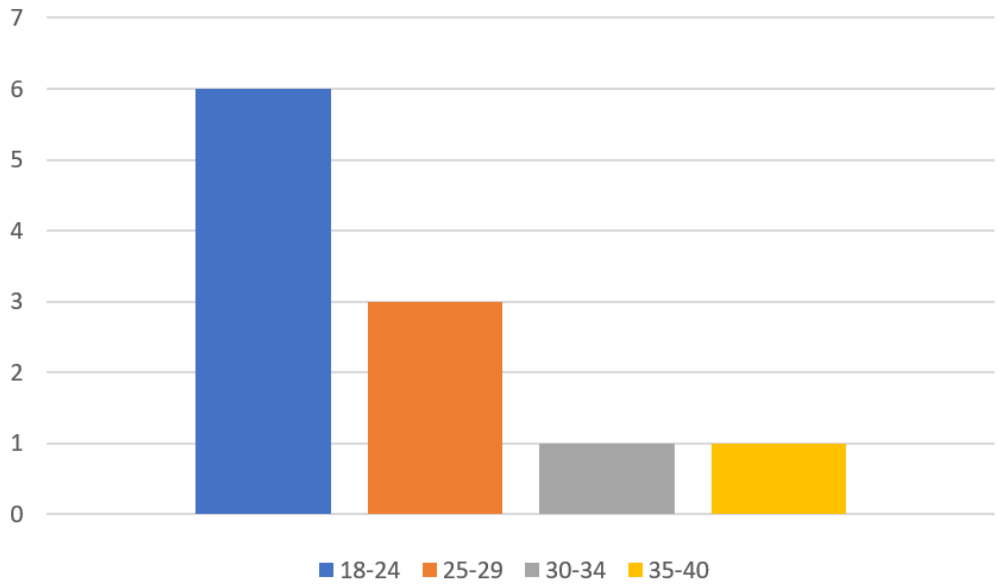

Abbildung 15: Altersgruppen der Probant*innen

In Bezug auf den Schulabschluss der Probant*innen hat ein Großteil mindestens die Fachholschulreife oder Allgemeine Hochschulreife. Zwei Probant*innen haben zu Ihrem Abschluss keine Angaben gemacht.

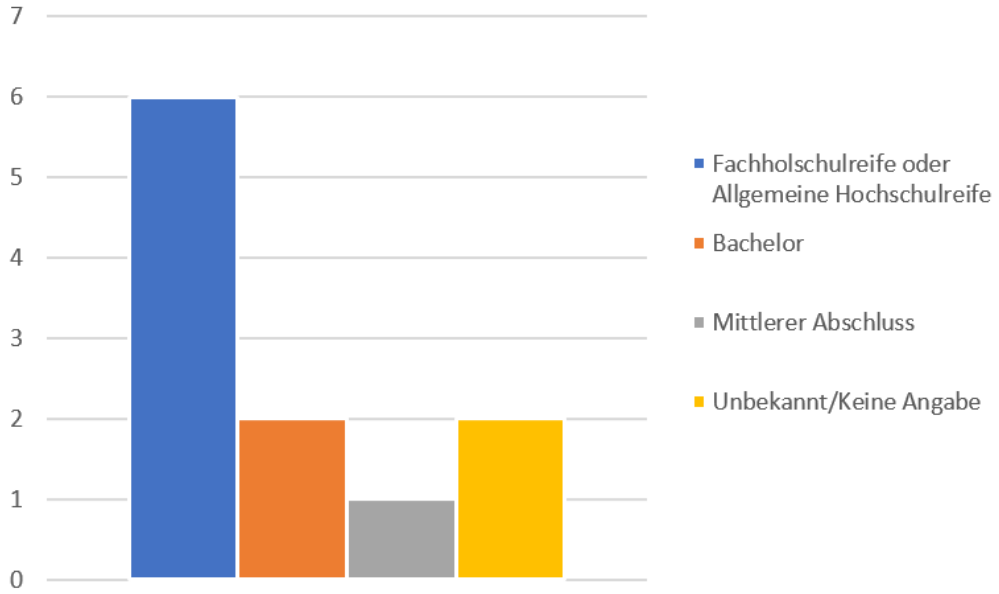

Abbildung 16: Höchste Schulabschlüsse der Probant*innen

Etwa die Hälfte der Probant*innen studieren, während die andere Hälfte verschiedenen Berufen nachgeht. Nur zwei Probant*innen Studieren Informatik, wir erhoffen uns hierdurch möglicherweise einen Blick auf die Software aus einer neutraleren Perspektive zu bekommen. Auch dies spiegelt sich in den Antworten zu der Frage *Wie würdest du deine Erfahrungen im Umgang mit Software bewerten* wieder. Hier haben die Probant*innen Ihre Erfahrungen auf einer Skala von 1 bis 5, wobei 1 für *Sehr unerfahren* und 5 für *Sehr erfahren* steht, im Durchschnitt mit 3,73 bewertet.

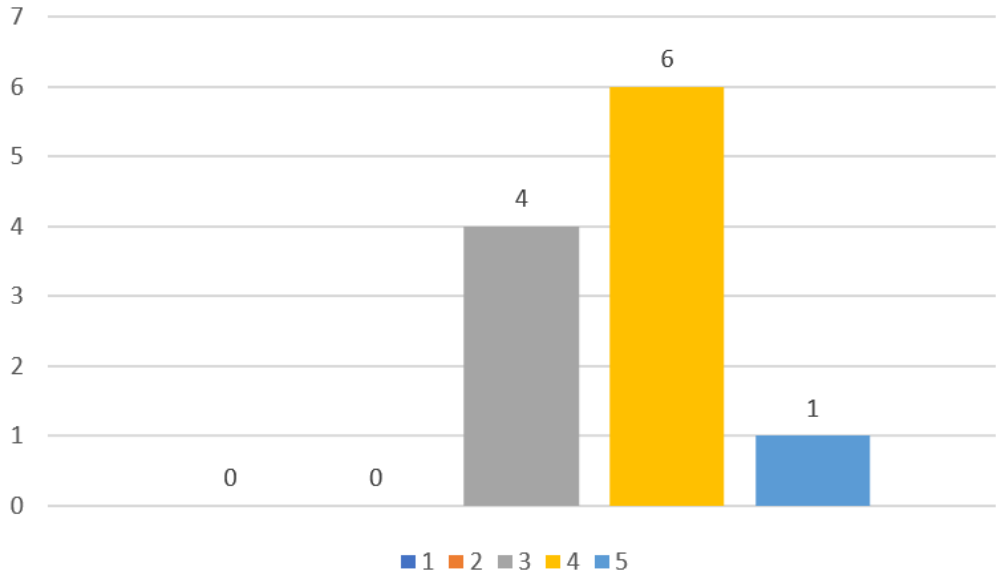

Abbildung 17: Antworten auf die Frage *Wie würdest du deine Erfahrungen im Umgang mit Software bewerten?* (Skala von 1-5, 1: Sehr unerfahren, 5: Sehr erfahren)

Unser Ziel, die Software auf möglichst vielen Browsern und Endgeräten zu testen, war bedingt erfolgreich. Das meistbenutzte Betriebssystem der Probant*innen war Windows 10, welches 45 % der Probant*innen benutzten. Von den Probant*innen, welche Windows 10 Benutzt haben, haben 60 % den Chrome-Browser von Google und 40 % den Firefox-Browser von Mozilla verwendet. Das am zweit meisten benutzte Betriebssystem war MacOS 11 mit 18 %. Die Probant*innen, die dieses Betriebssystem Benutzt haben, haben alle Firefox verwendet. Neben diesen Gruppen gab es noch eine Probant*in die Windows 11 und Firefox verwendet hat und eine Probant*in die iPadOS und Safari verwendet hat.

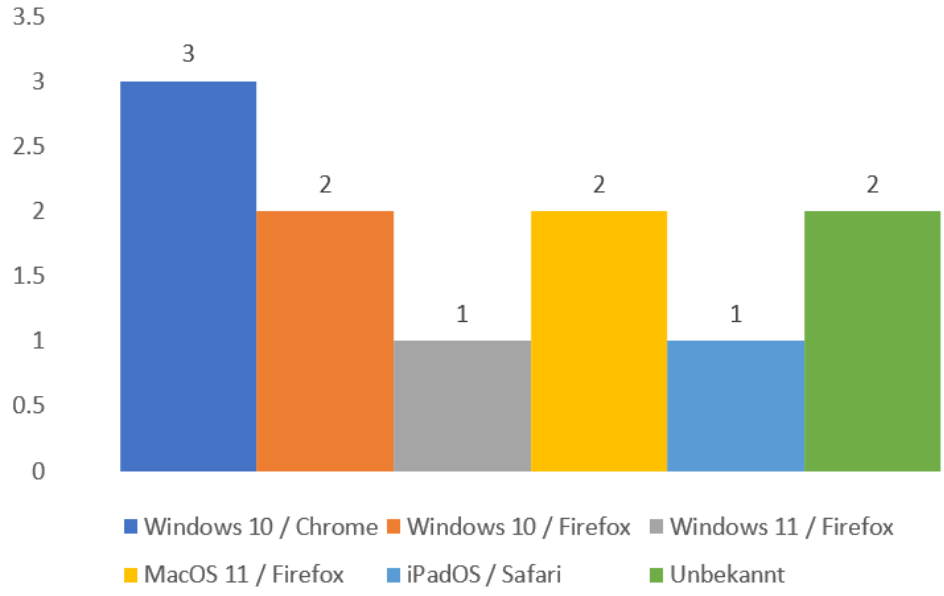

Abbildung 18: Ausführungsumgebungen der Probant*innen

Um die Korrektheit der Software zu analysieren, haben wir nach jedem Szenario gefragt ob alles funktioniert hat und wie Sicher Probant*innen waren, dass alles funktioniert hat. Auf einer Skala von 1 bis 5 (1: Sehr

unsicher, 5: Sehr sicher) wurde die Sicherheit, ob alles funktioniert hat im Durchschnitt von allen Szenarien mit 4,6 bewertet. Das Szenario *Server Löschen* hatte die höchste Bewertung mit 4,91, währen die Szenarien *Gameserver Verwaltung* und *Benutzer*innenverwaltung* die geringste Bewertung mit 4,36 hatten. Ein möglicher Grund hierfür wäre die Komplexität der Szenarien. In den beiden Szenarien mit der geringsten Bewertung sollten wesentlich mehr Schritte durchgeführt werden im Vergleich zu dem bestbewerteten Szenario. Diese Vermutung wird auch durch die anderen Szenarien unterstützt. Das Szenario *Gameserver Erstellen*, welches auch eher komplexer ist, wurde mit einer 4,4 bewertet, während die Szenarien, *Anmelden und Übersichtsseite* und *Benutzereinstellungen*, welche eher weniger komplex sind, jeweils mit einer 4,81 und einer 4,73 bewertet worden sind. Mit einer Standardabweichung von 0,25 zwischen den Werten gibt es aber keinen großen Unterschied zwischen den Szenarien.

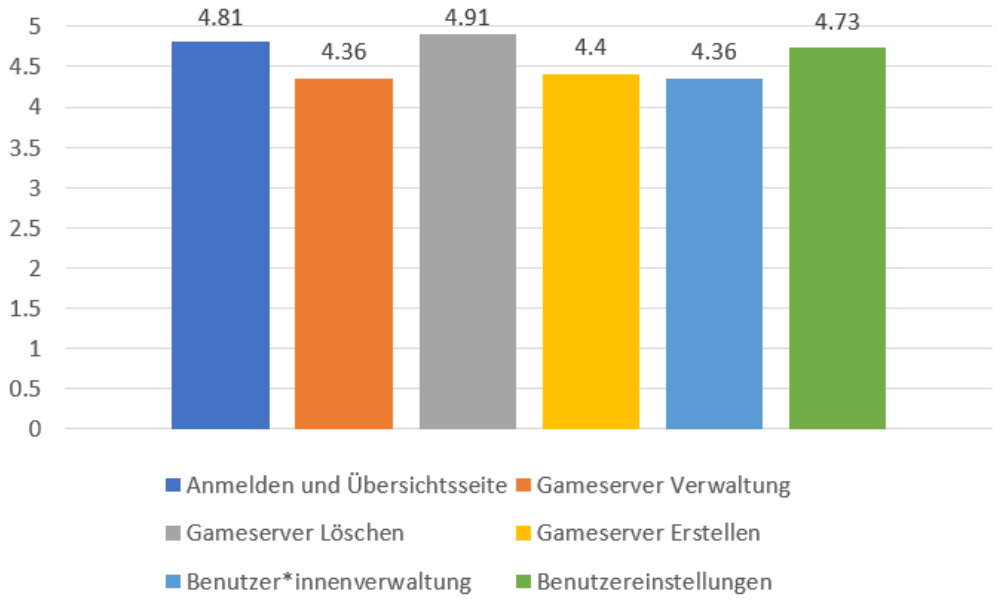

Abbildung 19: Bewertung, wie sicher sich die Probant*innen gefühlt hatten, ob alles funktioniert hat pro Szenario

Es gab nur eine Situation, in der etwas klar nicht funktioniert hat, dies war bei dem Szenario *Server Erstellen* bei der Probant*in, welche die Studie auf iPadOS und Safari bearbeitet hat. Hierbei konnten die Dateien, welche hochgeladen werden sollten, nicht ausgewählt werden. Dies ist durchaus nicht überraschend, da das Frontend nicht für mobile Endgeräte optimiert ist und auch während der Entwicklung nicht auf mobilen Endgeräten getestet wurde.

Die Gestaltung des Frontends wurde auf einer Skala von 1 bis 5 durchschnittlich mit 3,73 bewertet. Hieraus lässt sich schließen, dass hier Verbesserungsbedarf herrscht. Eine wesentliche Ursache für schlechtere Bewertungen war die schlechte Skalierung der Benutzeroberfläche auf kleineren Bildschirmen.

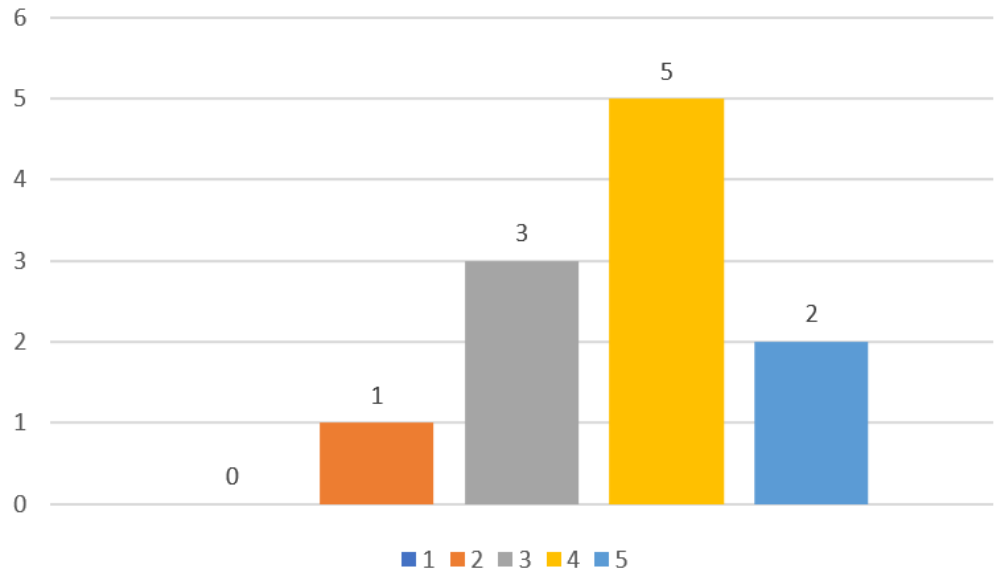

Abbildung 20: Antworten auf die Aussage *Ich fand das Tool visuell ansprechend* (Skala von 1-5, 1:Stimme gar nicht zu, 5: Stimme sehr zu)

Um die Benutzbarkeit zu analysieren, haben wir, wie bereits vorher erwähnt, die *System Usability Scale*, kurz SUS verwendet.

Der SUS-Score wird wie folgt berechnet: Für die Fragen 1, 3, 5, 7 und 9 wird von dem angegebenen Wert 1 abgezogen, für die Fragen 2, 4, 6, 8 und 10 rechnen wir 5 minus dem angegebenen Wert. Die errechneten Werte rechnen wir dann zusammen und multiplizieren das Ergebnis mit 2,5.[Bro95]

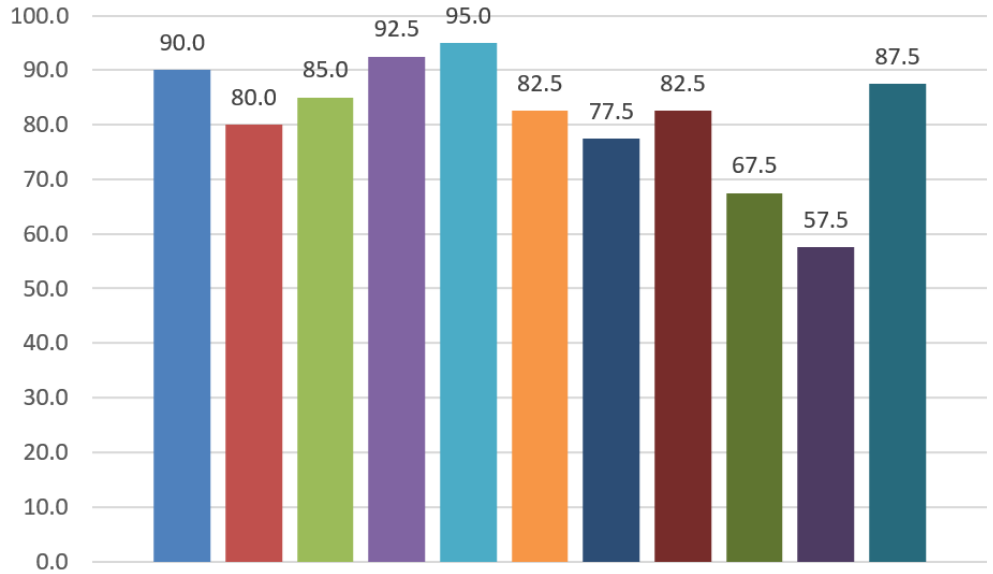

Abbildung 21: SUS-Score pro Probant*in

Interessant an den individuellen SUS-Scores der Probant*innen ist, dass der geringste SUS-Score von der ältesten Person stammt. Dies könnte darauf schließen, dass das Frontend für ältere Personen weniger Benutzbar ist. Um diese Aussage konkret zu analysieren, müssten hierzu aber mehr Daten vorliegen.

Um einen allgemeinen SUS-Score zu erlangen, berechen wir den Durchschnitt der einzelnen SUS-Scores. Der errechnete SUS-Score für das Frontend ist somit 81,6 (Standardabweichung: 11, Konfidenzintervall bei Konfidenzniveau von 95 %: 78,3-84,8). Im Vergleich zu dem, von Lewis und Sauro entwickelten Notensystem[LS18] würde dies dem 90-95 Perzentil bzw. der Note A entsprechen. Hieraus lässt sich schließen, dass die Benutzererfahrung bei verwenden des Frontends wesentlich überdurchschnittlich ist.

Grade	SUS	Percentile range
A+	84.1 – 100	96 – 100
A	80.8 – 84.0	90 – 95
A-	78.9 – 80.7	85 – 89
B+	77.2 – 78.8	80 – 84
B	74.1 – 77.1	70 – 79
B-	72.6 – 74.0	65 – 69
C+	71.1 – 72.5	60 – 64
C	65.0 – 71.0	41 – 59
C-	62.7 – 64.9	35 – 40
D	51.7 – 62.6	15 – 34
F	0 – 51.6	0 – 14

Abbildung 22: Sauro-Lewis Notenskala [LS18]

6.1.5 Kritik

Die Probant*innen unserer Benutzerstudie waren aus unserem Bekanntenkreis ausgewählt. Auch wenn wir versucht haben, Personen aus verschiedenen Gruppen mit verschiedenen Blickwinkeln zu finden, ist unsere Studie dennoch nicht repräsentativ für die allgemeine Gesellschaft. Möglicherweise wäre aber auch eine Auswahl der Probant*innen in die andere Richtung, also gezielt Probant*innen, welche die Software benutzen würden besser gewesen, da die Software letztendlich nicht von der Allgemeinheit, sondern von Softwareexperten verwendet werden soll. In beider Hinsicht wäre eine klarere Definierung der Zielgruppe besser gewesen.

Dadurch, dass die Personen aus unserem Bekanntenkreis kommen, könnte hier ein Bias vorhanden sein, welches die Werte verfälschen könnte.

Auch eine größere Anzahl an Probant*innen hätte helfen können, insbesondere bei Evaluation der Korrektheit. Hierbei wäre auch eine größere Bandbreite an Umgebungen (Betriebssystem, Browser, Bildschirmgröße)

hilfreich gewesen.

Da die SUS bereits bei einer geringen Anzahl an Probant*innen ein hilfreiches Ergebnis liefert,[Sau13] ist dies diesbezüglich weniger relevant, eine höhere Anzahl Probant*innen würde aber auch hier einen genaueren Wert liefern. Eine mögliche Verfälschung der Daten könnte aber aufgrund der eigenen Übersetzung der Fragen ins Deutsche zustande gekommen sein.

Das Verwenden der SUS bringt auch den Nachteil mit sich, dass diese keine Aussage über konkrete Schwachstellen in der Benutzbarkeit trifft. Um genau zu analysieren, was in Bezug auf Benutzbarkeit besser gemacht werden könnte oder was konkret gut gelaufen ist, wäre somit ein anderer Ansatz notwendig.

In Bezug auf die Evaluation des visuellen Eindrucks hätten neben einer größeren Anzahl an Probant*innen und einer größeren Bandbreite an Umgebungen eine Ausführlichere Befragung dazu helfen können, eine bessere Einsicht zu erlangen.

6.1.6 Fazit

Aus unserer Analyse lässt sich schließen, dass die Software in ihrer aktuellen Fassung bezüglich Benutzbarkeit oberhalb des Durchschnitts ist.

Bezüglich der Korrektheit lässt sich sagen, dass alle getesteten Funktionen in allen getesteten Umgebungen mit Ausnahme des Dateiuploads auf iPadOS mit Safari mit einer hohen Sicherheit funktionieren. Wobei hier aber eine höhere Anzahl an Probant*innen und eine größere Varianz der Umgebungen ein besseres Ergebnis geben könnten.

Über den visuellen Eindruck der Probant*innen bezüglich des Frontends konnten wir nur bedingt Schlüsse ziehen. Hier wäre eine umfassendere Befragung notwendig, um diesen Aspekt genauer zu analysieren.

6.2 Backend

Thorsten Friedewold:

In dem folgen Abschnitt wolle wir näher auf die Evaluation des Entwickelten Backend eingehen und dessen Funktionsfähigkeit testen.

6.2.1 Methodik

Um die Funktionalität des Backends zu testen, planen wir, das *Blackbox-Testing* anzuwenden. Dabei gehen wir davon aus, dass wir keine Informationen über die inneren Abläufe des Backends haben. Wir haben uns dazu entschlossen, da das Backend in Kombination mit dem Datenbank-Server, Dateispeicher-Server und Game-Servern viele Schnittstellen besitzt. Das isolierte Testen jeder einzelnen Komponente würde unserer Meinung nach nicht nur erheblich aufwendiger sein, sondern auch keinen so guten Einblick in das Zusammenspiel der einzelnen Komponenten ermöglichen.

6.2.2 Testumgebung

Um den Test transparent zu halten haben wir uns dazu einschlossen alle Dienste über ein *Docker-Compose-File* zu starten. Dabei werden alle Dienste (Dateispeicher, Datenbank, Frontend und Backend) in einer Datei zusammengestellt. Diese Datei kann dann auf verscheiden Computern, die *Docker* installiert haben ausgeführt werden. Wichtig dabei ist zu beachten, dass das Backend besondere Befugnisse beim Starten bracht, damit es weitere *Docker-Container* für die Game-Server starten kann.

6.2.3 Zielsetzung

Das Backend soll in der Lage sein, die in der Planung genannten Anforderungen zu erfüllen

6.2.4 Testen

Wir beginnen damit alle vom Backend bereitgestellten Anfragen zu sammeln um diese im Anschluss zu Testen.

Im folgenden testen wir die Funktionsfähigkeit des Backends. Den Test unterteilen wir in zwei Bereiche. Zunächst testen wir die Funktionsfähigkeit für die Verwaltung der Benutzer*innen. Dabei gehen wir die Möglichen Anfragen durch und überprüfen, ob das Backend die Änderungen dementsprechend verarbeitet.

Unser Test besteht aus einer Reihe von Anfragen an das Backend, dass alle Möglichen Anfragen abdeckt. Die Anfragen sind aufeinander Aufbauend, sodass der Ablauf nur durchlaufen werden kann, wenn alle Anfragen erfolgreich waren.

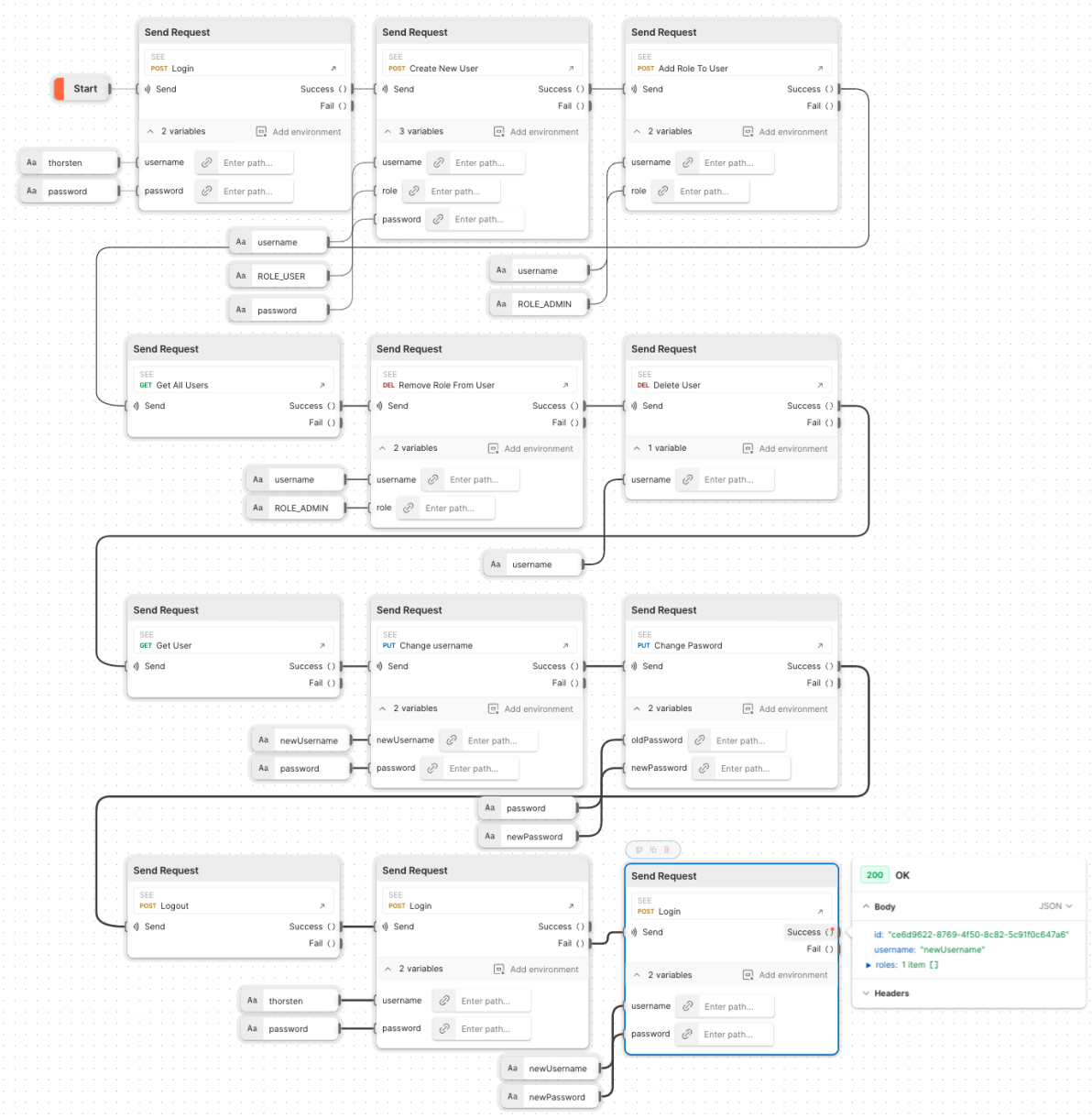

Abbildung 23: Test User

Zusammenfassung der Testsequenz:

- Anmelden

- Erstellen eines neuen Users
- Hinzufügen der Rolle *ADMIN* zum erstellten User
- Entfernen der Rolle *ADMIN* vom erstellten User
- Löschen des erstellten Users
- Ändern des Passworts des angemeldeten Users
- Ändern des Usernamens des angemeldeteten Users

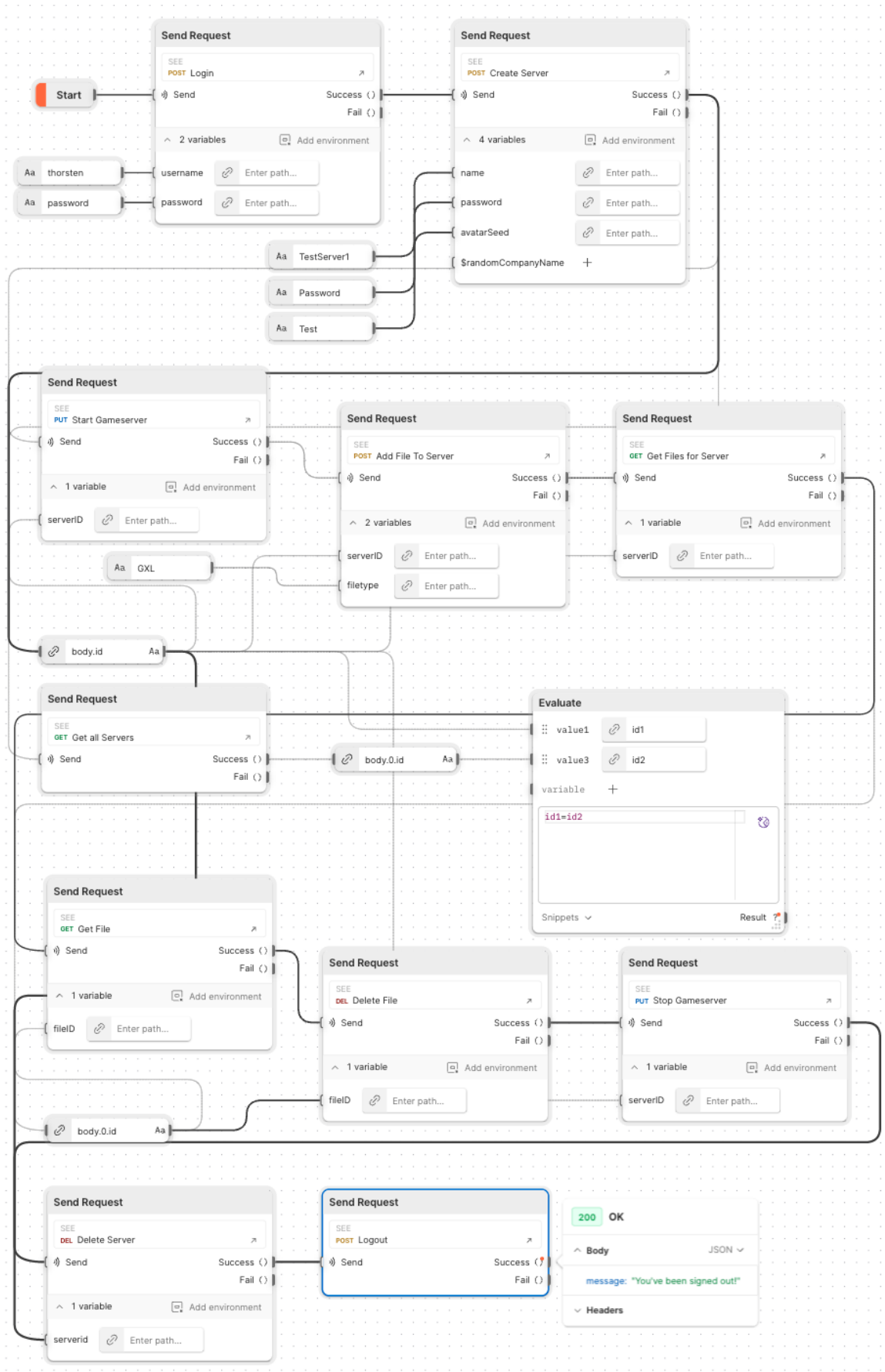

Abbildung 24: Testen der Server Anfragen an das Backend

Zusammenfassung der Testsequenz:

1. Anmelden
2. Erstellen eines neuen Servers
3. Starten des erstellten Servers
4. Hinzufügen einer Datei zu dem Game-Server
5. Abrufen der Dateien vom Game-Server
6. Überprüfen, ob die ID der Datei mit der, die dem Server hinterlegt ist übereinstimmt
7. Abrufen des Inhalts der hinzugefügten Datei
8. Löschen der Datei
9. Stoppen des Game-Servers
10. Löschen des Servers
11. Abmelden

Auch in diesem Fall konnten alle Anfragen Ordnungsgemäß ausgeführt werden. Weitere Tests wurden durch manuelle Überprüfung der Backend Logs und Datenbank vorgenommen.

6.2.5 Auswertung

Für die Auswertung beginnen wir damit uns alle Anfragen, die das Backend erlaubt anzuschauen und zu überprüfen, ob wir ihre Funktionsfähigkeit überprüfen konnten

Art	**Endpoint**	**Welche Dienste werden Getestet?**	**Getestet**	**Funktionsfähig**
GET	/server/	Backend; Datenbank	Ja	Ja
GET	/server/all	Backend; Datenbank	Ja	Ja
POST	/server/create	Backend; Datenbank	Ja	Ja
POST	/server/addFile	Backend; Datenbank; Dateispeicher	Ja	Ja
DELETE	/server/delete	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/server/files	Backend; Datenbank	Ja	Ja
GET	/user/me	Backend; Datenbank	Ja	Ja
GET	/user/all	Backend; Datenbank	Ja	Ja
POST	/user/create	Backend; Datenbank	Ja	Ja

POST	/user/ addRoleToUser	Backend; Datenbank	Ja	Ja
DELETE	/user/ removeRoleFromUser	Backend; Datenbank	Ja	Ja
DELETE	/user/delete	Backend; Datenbank	Ja	Ja
PUT	/user/ changeUsername	Backend; Datenbank	Ja	Ja
PUT	/user/ changePassword	Backend; Datenbank	Ja	Ja
POST	/user/signin	Backend; Datenbank	Ja	Ja
POST	/user/signout	Backend	Ja	Ja
GET	/file/get	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/file /client/csv	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/file /client/config	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/file /client/csv	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/file /client/-soruce	Backend; Datenbank; Dateispeicher	Ja	Ja
GET	/file /client/solution	Backend; Datenbank; Dateispeicher	Ja	Ja

Wie zu erkennen ist konnte das Backenend alle angebotenen Anfragen Ausführen. Die Entsprechenden Game-Server konnten gestartet und gelöscht werden. Die Hinzugefügten Dateien konnten dem Game-Server übergeben werden und waren in den Game-Servern verfügbar. Anzumerken ist, dass das Starten der Game-Server je nach System unterschiedlich lange dauern kann. Daher kann es dazu kommen, dass die Anfrage an das Backend einen Timeout bekommt. Dies bedeutet, dass die Bearbeitung zu viel Zeit in Anspruch genommen hat.

6.2.6 Stresstest

Abschließend wollen wir das Backend darauf prüfen, wie es auf viele Anfragen zur gleichen Zeit reagiert. Dafür haben wir drei Requests ausgewählt, um zu überprüfen, wie sich ihre Antwortzeiten ändern, wenn mehrere Nutzer*innen gleichzeitig Anfragen stellen. Für unseren Test verwenden wir die Requests *Login*, *Create Server* und *Delete Server*. Wir nutzen absichtlich nicht den *Start Server* Request, da das Starten der Game-Server extern über Docker geregelt wird. Ebenso testen wir nicht die Zeit für das Hochladen von Dateien, da sich das Ergebnis mit der Dateigröße ändern würde.

Alle Tests wurden mithilfe von Postman (einer Software zum Testen von APIs) durchgeführt, und die Tests liefen immer über einen Zeitraum von einer Minute. Die angegeben Zeit stellt den Durchschnitt dar

Art	Endpoint	Zeit für 1 Nutzer*in in ms	Zeit für 10 Nutzer*innen in ms	Zeit für 20 Nutzer*innen in ms
POST	/user/signin	166	118	225
POST	/server/create	14	15	29
DELETE	/server/delete	19	21	27

Unser Ziel war es, im Durchschnitt eine Ladezeit von unter 200 ms zu gewährleisten, da dies von den Nutzer*innen als *sofort* wahrgenommen wird [Res]. Dieses Ziel konnten wir zwar nicht vollständig zufriedenstellend implementieren, dennoch sind wir mit den Ergebnissen zufrieden.

6.3 Game-Server

Thorsten Friedewold:
In diesem Abschnitt wollen wir auf die Evaluation des Game-Servers eingehen. Wir halten zunächst die Anforderungen fest, die wir durch die von uns implementierten Maßnahmen testen wollen.

1. Allgemeine Funktionalität (Gameserver beitreten und verlassen)
2. Starten des Game-Servers aus dem Backend
3. Übernahme der an den Game-Server übergebenen Argumente (Raum-Passwort, etc.)
4. Echtzeit-Synchronisieren der ausgeführten Actions (Änderungen in der Spielumgebung, die zwischen den einzelnen Clients synchronisiert werden müssen)
5. Verzögertes Synchronisieren der ausgeführten Actions (Die Clients sollen alle Actions erhalten, die bereits vor ihrem Verbinden ausgeführt wurden.)
6. Herunterladen der Dateien, die bei der Erstellung des Game-Servers über das Frontend hinzugefügt wurden.

Im Folgenden wollen wir die aufgestellten Funktionalitäten testen.

- **Zu 1**
  Testaufbau: Wir starten zunächst eine Docker-Container Instanz unseres Game-Servers und versuchen anschließend, uns mit ihm zu verbinden. Wir geben dem Game-Server dabei kein Passwort, damit sich jeder verbinden kann.
  Test: Wir überprüfen, ob der Game-Server erstellt wurde und ob wir uns erfolgreich verbinden können.
  Erwartetes Verhalten: Der Game-Server wurde erstellt, und wir können uns erfolgreich verbinden.
  Tatsächliches Verhalten: Der Game-Server wurde erstellt, und die Verbindung war erfolgreich.
- **Zu 2**
  Testaufbau: Über das Frontend erstellen wir einen neuen Server im Backend, den wir anschließend starten.
  Test: Nun versuchen wir, uns mit einem Client wie im ersten Test zu verbinden.
  Erwartetes Verhalten: Der Game-Server wurde erstellt, und wir können uns mit dem Client verbinden.
  Tatsächliches Verhalten: Der Game-Server wurde erstellt, und die Verbindung mit dem Client war erfolgreich.
- **Zu 3**
  Testaufbau: Wie beim Test 2 erstellen wir über das Frontend einen neuen Game-Server.
  Test 1: Dieses Mal setzen wir jedoch ein Passwort beim Erstellen und Starten des Game-Servers. Nun

versuchen wir, uns wieder mit einem Client zu verbinden.
Test 2: Nun geben wir beim Client das richtige Passwort an und versuchen erneut, uns zu verbinden.
Erwartetes Verhalten 1: Der Client kann sich nicht verbinden, und der Game-Server lehnt die Verbindung ab.
Tatsächliches Verhalten 1: Der Game-Server lehnt die Verbindung ab.
Erwartetes Verhalten 2: Der Client kann sich verbinden, und der Game-Server akzeptiert die Verbindung.
Tatsächliches Verhalten 2: Der Game-Server akzeptiert die Verbindung.

- **Zu 4**
  Testaufbau: Wir starten den Game-Server (ohne Passwort) und verbinden uns dann mit zwei Clients.
  Test: Wir verschieben, rotieren, skalieren, etc. nun einige Objekte im virtuellen Meetingraum und überprüfen, ob sich diese Änderungen auf dem anderen Client synchronisieren. Zusätzlich überprüfen wir die Nachrichten, die der aktive Client versendet und welche Nachrichten der passive Client erhält.
  Erwartetes Verhalten: Beide Clients sollten die Objekte an den gleichen Stellen sehen und die gleichen *Player-Actions* aufgeführt haben.
  Tatsächliches Verhalten: Beide Clients stellen die Objekte gleich dar und haben die gleichen *Player-Actions* aufgeführt.

- **Zu 5**
  Testaufbau: Wir erstellen einen Game-Server (ohne Passwort) und verbinden uns mit einem Client.
  Test: Wir verschieben, rotieren, skalieren, etc. wie im Test 4 verschiedene Objekte und trennen dann die Verbindung. Anschließend verbinden wir den Client erneut mit dem Game-Server.
  Erwartetes Verhalten: Die Änderungen, die wir mit dem Client vorgenommen haben, sollten nach erneuter Verbindung bestehen bleiben.
  Tatsächliches Verhalten: Die Änderungen des Clients bleiben bestehen.

- **Zu 6**
  Testaufbau: Wir erstellen einen Game-Server (mit einem Passwort) und hinterlegen beim Erstellen im Frontend verschiedene Dateien. Danach verbinden wir uns über einen Client.
  Test: Wir navigieren zum Verzeichnis *Multiplayer* im Client und überprüfen, ob die hinterlegten Dateien heruntergeladen wurden. Zusätzlich kontrollieren wir, ob die hochgeladene *SOURCE*-Datei entpackt wurde.
  Erwartetes Verhalten: Alle hochgeladenen Dateien sind vorhanden, und die Zip-Datei wurde entpackt.
  Tatsächliches Verhalten: Alle hochgeladenen Dateien sind vorhanden, und die Zip-Datei wurde entpackt.

Wir konnten durch unsere Tests evaluieren, dass der Game-Server den von uns gestellten Anforderungen genügt und in Kombination mit unserem Frontend und Backend innerhalb eines Docker-Containers verwendet werden kann.

7 Fazit

Simon Leykum:

Insgesamt konnten einen Großteil der anfangs definierten angestrebten Funktionalitäten umsetzen. Aufgrund des Umfangs der implementierten Funktionalitäten konnten wir leider nicht das Speichern des aktuellen Stands einer Code-City innerhalb eines Meetingraums implementieren. Abgesehen davon konnten wir alles umsetzen.

Den größten Aufwand bezüglich des Entwickelns des Game-Servers mussten wir bei dem Umstellen der Netzwerkstruktur treiben. Hierbei mussten wir uns zunächst in die aktuelle komplexe Netzwerkstruktur

einarbeiten und diese dann versuchen zu vereinfachen und zu vereinheitlichen. Mit unserer Lösung diesbezüglich sind wir aber sehr zufrieden, da wir der Meinung sind, dass hierdurch die Netzwerkstruktur um einiges vereinfacht wurde.

Auch das *Deployment* hat für einen hohen Aufwand gesorgt. Das Testen während der Entwicklung war hier aufgrund der hohen Anzahl an Faktoren besonders aufwendig. Auch hier hat sich der Aufwand unserer Ansicht nach sehr gelohnt, da hierdurch das Starten und Ausliefern unserer Lösung wesentlich vereinfacht wurde.

Wir sind zudem sehr über das Ergebnis unserer Benutzerstudie erfreut, da wir hier eine Bestätigung dafür bekommen haben, dass wir unsere gesetzten Ziele bezüglich der Entwicklung des Frontends erreichen konnten.

Auch die Evaluation unseres Backends und unseres Game-Servers ist positiv gelaufen und zeigt, dass wir unsere gesetzten Ziele durchsetzen konnten.

Zusammenfassend können wir sagen, dass wir mit dem Ergebnis unserer Arbeit sehr zufrieden sind und hierbei viel lernen konnten.

8 Ausblick

Aufgrund unserer Evaluationen und einigen Überlegungen möchten wir nun ein paar Weiterentwicklungsmöglichkeiten bezüglich der entwickelten Komponenten und Möglichkeiten für weitere Evaluation und Vertiefungen darstellen.

8.1 Frontend

Simon Leykum:

Ein wesentlicher Aspekt der Weiterentwicklung des Frontends ist die Anpassbarkeit der Benutzeroberfläche für verschiedene Bildschirmgrößen. Dies ist insbesondere wichtig, falls das Frontend in Zukunft für mobile Endgeräte optimiert werden soll. Hier wäre es auch nötig, die Funktionalität auf diesen genauer zu überprüfen. Auch das Design des Frontends könnte tiefer analysiert und möglicherweise angepasst werden. Des weiteren könnte eine Studie angefertigt werden, um genaue Verbesserungsmöglichkeiten bezüglich der Benutzbarkeit zu analysieren.

8.2 Backend

Thorsten Friedewold:

Im Hinblick auf das Backend bestehen verschiedene Möglichkeiten zur Weiterentwicklung, insbesondere im Bereich der Skalierbarkeit. Ein Beispiel hierfür wäre eine Implementierung von *JWT*, die es theoretisch ermöglichen würde, das Backend in einem skalierbaren Cluster zu betreiben. Allerdings wäre dafür eine Auslagerung der Game-Server-Verwaltung erforderlich. Ein alternativer Ansatz zur Weiterentwicklung liegt in der Verbesserung der Game-Server-Verwaltung. Der aktuelle Stand dieser Implementierung ist recht simpel und könnte bei längerem Betrieb anfällig für Fehler sein. Zusätzlich wäre es denkbar, weitere Anpassungsoptionen zu integrieren. Beispielsweise könnten verschiedene Szenarien für die Game-Server gespeichert oder Einladungs-E-Mails an Personen versendet werden.

8.3 Game-Server

Simon Leykum:

Im Bezug auf die Implementierung des Synchronisierens der Player-Actions mit einem sich neu verbindenden Clients gibt es noch einige Verbesserungsmöglichkeiten. Da Listen in C# eine Maximalgröße haben, wird das Speichern der ausgeführten Player-Actions in einer Liste irgendwann zu einem Problem führen. Eine Möglichkeit, dies vorzubeugen, wäre es, die Liste regelmäßig zu leeren. Entweder könnte grundsätzlich die Liste nach einer bestimmten Zeit, oder, z. B. wenn alle Clients sich von dem Server getrennt haben, geleert werden. Wenn zusätzlich die Funktion implementiert wird, dass ein Zwischenstand der Szene gespeichert werden kann, könnte dies das Problem lösen. Eine weitere Lösung wäre eine Umänderung der aktuellen Player-Action-Struktur, sodass die GameObjects, welche die Code-City darstellen, direkt mithilfe von *Netcode for GameObjects* synchronisiert werden.

Dieser Ansatz könnte möglicherweise auch auf einen großen Anteil der Player-Actions angewandt werden, sodass diese möglicherweise komplett ersetzt werden könnten. Ob dieser Ansatz vorteilhaft oder umsetzbar ist müsste aber genauer evaluiert werden.

Um die Entwicklung des Game-Servers zu finalisieren, wäre zudem eine tiefere Analyse der Funktionalität notwendig.

Literatur

[Bro95] J. Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (Nov. 1995).

[Con] *Passing Data Deeply with Context*. URL: https://react.dev/learn/passing-data-deeply-with-context.

[Kos23] R. Koschke. *SEE*. 2023. URL: https://github.com/uni-bremen-agst/SEE.

[Lat] *MonoBehaviour.LateUpdate()*. 2023. URL: https://docs.unity3d.com/2023.2/Documentation/ScriptReference/MonoBehaviour.LateUpdate.html.

[LS18] J. Lewis und J. Sauro. "Item Benchmarks for the System Usability Scale". In: *Journal of User Experience* 13 (Mai 2018), S. 158–167.

[Lub22] S. Luber. *Was ist JSON Web Token (JWT)?* 2022. URL: https://www.security-insider.de/was-ist-json-web-token-jwt-a-1094265/.

[Neta] *Back to the 'open source'*. 2016. URL: https://networkcomms.net/back-to-the-open-source/.

[Netb] *Networking made easy, out of the box*. 2014. URL: https://networkcomms.net/.

[Nfga] *Netcode for GameObjects Changelog*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/release-notes/ngo-changelog/.

[Nfgb] *NetworkBehaviour*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkbehavior/.

[Nfgc] *NetworkObject. Ownership*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/#ownership.

[Nfgd] *NetworkObject*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/.

[Nfge] *Sending Events with RPCs*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/.

[Nfgf] *ServerRpc*. 2023. URL: https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/serverrpc/.

[Rea] *Meet the Team*. URL: https://react.dev/community/team.

[Res] *Response Time*. URL: https://sematext.com/glossary/response-time/.

[Sau13] J. Sauro. *10 Things to Know About the System Usability Scale (SUS)*. Juni 2013. URL: https://measuringu.com/10-things-sus/.

[Uni] *Unity Standalone Player command line arguments*. 2023. URL: https://docs.unity3d.com/2023.2/Documentation/Manual/PlayerCommandLineArguments.html.

[Wha] *What is Figma?* URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma-.

Anhang

Zu dieser Arbeit wird ein USB-Stick mit folgenden Dateien angehangen:

- **Bachelorarbeit** Ein Ordner mit folgenden Dateien
  - **bachelorarbeit.pdf** Diese Arbeit in digitaler Form
  - **data.xlsx** Die Rohdaten der Benutzerstudie
  - **Bilder** Ein Ordner mit allen in dieser Arbeit verwendeten Bildern in Originalauflösung
- **Gameserver-Linux** Ein Ordner mit einer unter Linux ausführbaren Variante unseres Gameservers
- **Gameserver-Windows** Ein Ordner mit einer unter Windows ausführbaren Variante unseres Gameservers
- **SEE-Client** Ein Ordner mit einer unter Windows ausführbaren Variante von SEE
- **Backend** Ein Ordner mit dem Quellcode des Backends
- **Frontend** Ein Ordner mit dem Quellcode des Frontends
- **compose.yaml** Eine Datei um unsere Lösung mithilfe von Docker auszuführen
- **readme.md** Eine Datei die erklärt wie unsere Lösung mithilfe von Docker ausgeführt werden kann